

Output Constraints in Multimedia Database Systems

Dissertation zur Erlangung des akademischen Grades
Doktor–Ingenieur (Dr.-Ing.)

vorgelegt der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von Dipl.–Inf. Thomas Heimrich

1. Gutachter: Prof. Dr. habil. Kai-Uwe Sattler
2. Gutachter: Prof. Dr. habil. Klaus Meyer-Wegener
3. Gutachter: Prof. Tamer Özsu

Tag der Einreichung: 19.09.2005
Tag der wissenschaftlichen Aussprache: 13.02.2006

urn:nbn:de:gbv:ilm1-2006000019

Zusammenfassung

Semantische Fehler treten bei jeder Art von Datenverwaltung auf. Herkömmliche Datenbanksysteme verwenden eine Integritätskontrolle, um semantische Fehler zu vermeiden. Um die Integrität der Daten zu gewährleisten werden Integritätsregeln benutzt. Diese Regeln können allerdings nur die Konsistenz einfach strukturierter Daten überprüfen.

Multimedia Datenbanksystem verwalten neben einfachen alphanumerischen Daten auch komplexe Mediendaten wie Videos. Um die Konsistenz dieser Daten zu sichern, bedarf es einer erheblichen Erweiterung des bestehenden Integritätskonzeptes. Dabei muss besonders auf die konsistente Datenausgabe geachtet werden. Im Gegensatz zu alphanumerischen Daten können Mediendaten während der Ausgabe verfälscht werden. Dieser Fall kann eintreten, wenn eine geforderte Datenqualität bei der Ausgabe nicht erreicht werden kann oder wenn Synchronisationsbedingungen zwischen Medienobjekten nicht eingehalten werden können. Es besteht daher die Notwendigkeit, *Oupput Constraints* einzuführen. Mit ihrer Hilfe kann definiert werden, wann die Ausgabe von Mediendaten semantisch korrekt ist. Das Datenbanksystem kann diese Bedingungen überprüfen und so gewährleisten, dass der Nutzer semantisch einwandfreie Daten erhält.

In dieser Arbeit werden alle Aspekte betrachtet, die notwendig sind, um Ausgabebedingungen in ein Multimedia Datenbanksystem zu integrieren. Im einzelnen werden die Modellierung der Bedingungen, deren datenbankinterne Repräsentation sowie die Bedingungsüberprüfung betrachtet.

Für die Bedingungsmodellierung wird eine *Constraint Language* auf Basis der Prädikatenlogik eingeführt. Um die Definition von zeitlichen und räumlichen Synchronisationen zu ermöglichen, verwenden wir Allen-Relationen. Für die effiziente Überprüfung der Ausgabebedingungen müssen diese aus der Spezifikationssprache in eine datenbankinterne Darstellung überführt werden.

Für die datenbankinterne Darstellung werden *Difference Constraints* verwendet. Diese erlauben eine sehr effiziente Bedingungsüberprüfung. Wir haben Algorithmen entwickelt, die eine effiziente Überprüfung von Ausgabebedingungen erlauben und dies anhand von Experimenten nachgewiesen. Neben der Überprüfung der Bedingungen müssen Mediendaten so synchronisiert werden, dass dies den Ausgabebedingungen entspricht. Wir haben dazu das Konzept des *Output Schedules* entwickelt. Dieser wird aufgrund der definierten Ausgabebedingungen generiert.

Durch die Ausgabebedingungen, die in dieser Arbeit eingeführt werden, werden semantische Fehler bei der Verwaltung von Mediendaten erheblich reduziert. Die Arbeit stellt daher einen Beitrag zur qualitativen Verbesserung der Verwaltung von Mediendaten dar.

Abstract

Semantic errors exist as long as data are managed. Traditional database systems try to prevent this errors by proposing integrity concepts for stored data. Integrity constraints are used to implement these integrity concepts. However, integrity constraints can only detect semantic errors in elementary data.

Multimedia database systems manage elementary data as well as complex media data, like videos. Considering these media data we need a much wider consistency concept as traditional database systems provide. Especially, data output of media data must be taken into account. In contrast to alphanumeric data the semantics of media data can be falsified during data output if data quality or synchronization of data are not suitable. Thus, we need a concept for *output constraints* that allow for preventing semantic errors in case of data output. For integrating output constraints into a multimedia database system we have to consider *modelling*, *representation* and *checking* of output constraints.

For modelling output constraints we have introduced a constraint language which uses the same principles as traditional constraint languages. Our constraint specification language must support temporal and spatial synchronization constraints. However, it is desired to support both kinds of synchronization in almost the same manner. Therefore, we use Allen-Relations for defining temporal synchronization constraints as well as for defining spatial synchronization constraints.

We need a database internal representation of output constraints that makes efficient constraint checking possible. The Allen-Relations used in the constraint language cannot be checked efficiently. However, difference constraints are a class of constraints that allows an very efficient checking. Therefore, we use difference constraints as database internal representation of output constraints.

As methods for checking consistency of output constraints we use an approach based on graph theory as well as an analytical approach. Both approaches require a constraint graph as data structure. For data output we need an output order that is adequate to the defined output constraints. This ‘output schedule’ can be produced based on the output constraints.

With output constraints, proposed in this thesis, semantical correctness of media data considering the data output can be supported. Thus, the contribution of this work is an qualitative improvement of managing media data by database systems.

Acknowledgements

My thanks aim to all people who supported my work. Kai-Uwe Sattler, my supervisor, gave me the freedom to develop my own ideas and challenged them in numerous discussions. Furthermore, he gave me the chance to write research papers and helped me with many hints. Actually, Klaus Meyer-Wegener was my supervisor as well. He helped me very much during the hard first time of my research. Furthermore, he gave me the possibility of visiting him regularly which was always very useful. Thanks go to Tamer Özsu. He gave me the opportunity to visit him and his research group at the University of Waterloo. I am very glad that he accepted to review this thesis. My special thanks go to Katja Hose. She cleans up the English, found some content errors, and made the thesis readable.

Contents

1	Motivation	1
2	Requirements on Output Constraints	6
2.1	Wave Field Synthesis	6
2.2	Virtual University	11
2.3	Hospital Database Application	13
2.4	Conclusion	15
3	Related Work	17
3.1	Data Models and Architecture of MMDBS	17
3.1.1	Requirements on a Multimedia Database System	17
3.1.2	Data Models for Multimedia Database Systems	19
3.1.3	Architecture of Multimedia Database Systems	20
3.1.4	Existing Approaches for Multimedia Database Systems	22
3.2	Integrity Constraints	23
3.2.1	Types and Semantics of Integrity Constraints	24
3.2.2	Specification of Integrity Constraints	26
3.2.3	Checking of Integrity Constraints	28
3.3	Presentation of Media Data	30
3.3.1	Quality of Service	30
3.3.2	Presentation Models for Multimedia Data	31
3.4	Consistency Rules for Multimedia Data	35
3.4.1	Constraints on Data Quality	35
3.4.2	Temporal Consistency	36
3.4.3	Spatial Consistency	38
3.4.4	Constraints in Commercial Database Systems	39
3.5	Conclusion	39
4	Modelling of Output Constraints	41
4.1	Taxonomy of Output Constraints	41
4.2	A Classification of Output Constraints	45
4.3	Fundamentals of Output Constraints	46
4.3.1	Elements of Constraint Notation	46

4.3.2	Temporal Logic	47
4.3.3	Specification of Temporal Constraints	49
4.3.4	Specification of Spatial Constraints	50
4.4	The Database Output	53
4.5	General Structure of Output Constraints	54
4.6	Specification Language for Output Conditions	55
4.6.1	Conditions for Static Output Parameters	56
4.6.2	Conditions for Dynamic Output Parameters	57
4.6.3	Synchronization Conditions	61
4.7	Notation of Output Constraints	65
4.8	Conclusion	67
5	Internal Representation of Output Constraints	69
5.1	Process of Output Constraint Management	69
5.2	Representation of Output Constraints	72
5.2.1	Output Constraints on Output Parameters	72
5.2.2	Output Constraints for Synchronization	73
5.2.3	Transforming Temporal and Spatial Lengths	77
5.3	Storage Structure of Output Constraints	78
5.4	Conclusion	79
6	Output Constraints and Data Consistency	80
6.1	Execution Model of Output Constraints	81
6.2	Consistency Check using Graph Theoretical Approach	82
6.3	Building a Partial Constraint Graph	84
6.3.1	Restrictions on Constraint Graphs	84
6.3.2	Modifications of Allen-Relations	85
6.3.3	Modifications of Output Objects	90
6.3.4	Algorithm for Building a Partial Graph	92
6.4	Consistency Check using Analytic Approach	94
6.5	Auxiliary Data Structure for Checking Consistency	95
6.6	Conclusion	96
7	Management of Output Schedules	97
7.1	Use Cases for Output Schedules	97
7.2	Producing an Output Schedule	98
7.3	Structure of Output Schedules	100
7.4	Consistency of Output Schedules	101
7.4.1	Modifying Allen-Relations in the Front Section of the Output Line	101
7.4.2	Modifying Allen-Relations at the End of the Output Line . . .	104
7.4.3	Modifications on Output Objects	105
7.4.4	Output Schedules with Several Output Lines	105

7.5	Checking Output Consistency During the Output Process	108
7.6	Conclusion	109
8	Implementation and Evaluation	110
8.1	Output Constraints for Wave Field Synthesis	110
8.1.1	Modelling of Output Constraints	111
8.1.2	Integration of Output Constraints into a Database System . . .	114
8.1.3	Checking Output Constraints	114
8.1.4	Producing Data Output	115
8.2	Implementation of Further Scenarios	117
8.3	Experiments	117
8.4	Conclusion	123
9	Conclusion and Future Work	125
A	Harmless and Critical Allen-Relations	129
B	Analytical Constraints for Allen-Relations	140

Chapter 1

Motivation

A main reason for using databases for managing data is storing data in a semantical correct manner. *Semantic errors* occur if stored data do not fit the part of the real world that is modeled by the database. Mostly, semantic errors arise from incorrect data input. Sometimes a user does not have enough knowledge about the problem domain that the database was built for. This might result in creates incorrect data input unintentionally.

Whether or not data are semantically correct depends on the specific problem domain. We can check the semantic consistency of data either in the database application or inside the database system.

To prevent semantic errors, database systems possess an *integrity concept*. This includes a descriptive constraint language for defining semantic integrity constraints as well as efficient checking methods for integrity constraints [GA93]. It is generally accepted that checking semantic consistency should be done inside the database system because it has the following advantages:

- The integrity constraints can be defined in a declarative manner.
- The development of database applications becomes easier because checking data consistency does not have to be considered.
- Integrity constraints are checked for ad-hoc queries as well as for database queries generated by database applications.
- The database system can execute an optimized constraint checking.

Integrity constraints supported by traditional database systems are usually classified into *static* and *dynamic* integrity constraints. Static integrity constraints work just on one database state. Thus, their checking is relatively easy. Furthermore, defining these integrity constraints is very intuitive. Usually, check-clauses are used for specifying this group of integrity constraints. Dynamic Integrity constraints are more complex because several database states are required for checking them. Furthermore,

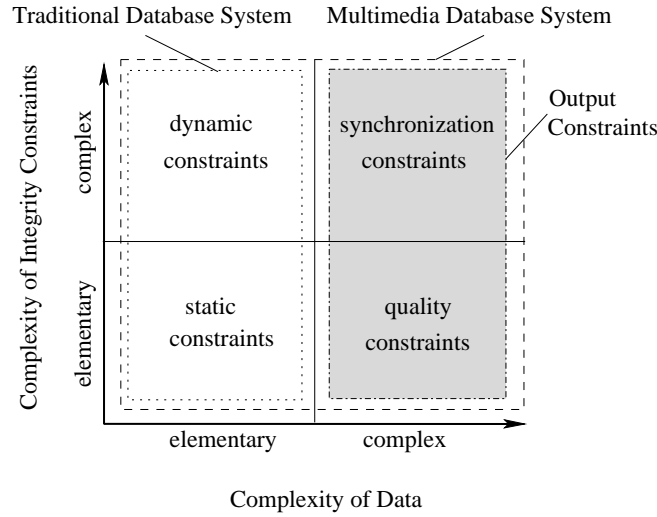


Figure 1.1: Classification of Integrity Constraints

triggers or assertions are used to define dynamic integrity constraints. However, both kinds of integrity constraints work on elementary data (figure 1.1).

New developments in several commercial and scientific fields make greater demands on database systems. Take as an example the data management of a hospital. It is no more sufficient to manage only alphanumeric data of patients. We also have to manage media and multimedia data which arise from several modern diagnostic techniques.

Multimedia Database Systems are built to manage structured data as well as media data, like videos, audios, images, and full text. These systems must be able to manage large amounts of multimedia data. Some special architectures have been proposed [BBH⁺02, Ber02] which allow efficient storage, retrieval, presentation, and manipulation of multimedia data.

As in traditional database systems, correctness of stored data is of great importance in a multimedia database system. Thus, a multimedia database system must support the same database techniques in order to prevent errors as traditional database systems do. The characteristics of media data are different from alphanumeric data. Therefore, new concepts for maintaining data semantics are required and must be integrated into a multimedia database system.

We want to use the presentation of stored media data as an example to clarify these new requirements demanded by a multimedia database system. Usually, media data are stored in a database but a presentation script outside the database is used to define the presentation constraints. In this case errors can arise from the following:

- Some media data are physical signals. Therefore, physical parameters are important for the semantics of this kind of data. Usually, these physical parameters must be into specific ranges during the data output process otherwise the semantics of the data can be disordered. For the user there is no difference between

data which are stored incorrectly or those which are stored correctly but output in an incorrect manner.

It is assumed that we define an output of audio data using a certain sample rate. We make this definition in our presentation script. If the data output is done with an incorrect sample rate, the semantics of the audio will be incorrect, too. This means the audio can be heard but the content cannot possibly be understood. However, the script cannot detect whether the sample rate is wrong or not because the database system does not provide this information.

However, it is hard to guarantee an end to end data quality for the whole data output process but the first step to guarantee a certain data quality must be done in the multimedia database system. If we observed the sample rate of an audio during the output process we would have the basis for preventing semantic errors.

- Multimedia documents consist of different (multi-)media objects and relationships between them. In our presentation script we can define a film that consists of an audio stream and a video stream connected by synchronization constraints. Often, the output of these streams must be lip synchronous. The semantics of the multimedia document will be distorted if the synchronization relationships cannot be fulfilled.

Here we have the same problem as in the case above. The database application cannot detect whether the synchronization constraint is violated. A constraint violation can be caused by delays or other errors during the data output or by storing media data which cannot fulfill the defined synchronization constraints. For instance, a lip synchronized output of our defined film is only possible when audio and video stream have the same length.

As we can see from the points above semantically incorrect data can appear during the data output. The reason for these semantic errors are either falsifications of special physical properties or incorrect synchronizations of media data. These two classes of integrity constraints are completely ignored in traditional database systems because they are not critical for alphanumeric data. However, they are important to ensure the semantics of media data.

We have enhanced the traditional classification of integrity constraints, as shown in figure 1.1. Usually, constraints on output parameters define requirements for quality of media data (e.g. a certain sample rate). Therefore, we call these constraints *quality constraints* in figure 1.1. Quality constraints are easier than synchronization constraints because output parameters are usually stored as metadata of media objects. The resolution of an image is an example of such metadata. We can use traditional comparing operators (e.g. $<$, $>$, $=$) to define constraints on these output parameters. However, the checking of these constraints must be done during the data output process. Thus, we need new constraint checking concepts which are beyond the means of usual concepts.

Synchronization constraints are more complex because we need possibilities for specifying as well as for checking several classes of synchronization constraints. We have to consider synchronization constraints for temporal, spatial, and spatiotemporal synchronizations.

Synchronization constraints as well as quality constraints are used to define requirements for the multimedia data output. Therefore, we build a generalized constraint class called *output constraints* (figure 1.1) which contains both aspects. Output constraints do not take the database state into account like integrity constraints do, rather they restrict *output objects*. Output objects are built of stored media data objects. They exist only during data output. As an example, an output object for a stored text could be that text as image (PDF). The same text can be transformed into an audio for data output. In this case the audio is the output object. To satisfy *output consistency* all defined output constraints associated with an output process must be fulfilled. As we will see the output consistency is not independent from consistency defined by means of integrity constraints. In some cases integrity constraints can be derived from output constraints. Thus, both types of constraints are related.

This work supports the statement that integration of output constraints into a multimedia database system is necessary. The following points demonstrate the advantages of such an approach:

- The producer of multimedia data can define output constraints in order to support the correct semantics of the data. The multimedia database systems guarantees the desired semantics of multimedia data for each data output.
- Output constraints can be checked before or during the data output. Only those data are sent to the application which fulfill the specified output constraints. If output constraints are violated during the output process a defined action is executed.
- The database can guarantee that the stored data are suitable for the defined output constraints. Only such media data are stored that fulfill the defined output constraints.
- By using output constraints for synchronization the database system can build an output order for output objects. This order is called *output schedule*. The database system can use the output schedule for a data organization that allows for an efficient data output.
- Usually, values for output parameters are also defined in database queries. The multimedia database system can check if these values can be tolerated considering the semantics of the multimedia data.

The Contribution of this Thesis

In this work we want to introduce concepts for handling output constraints inside a multimedia database system. The first step is defining the semantics of output constraints. This means we must determine types of output constraints and their utilization. Therefore, chapter 2 analyses several application scenarios especially, considering requirements for output constraints. Based on our requirement analysis chapter 3 gives an overview of related works which can be used as basis for our output constraints.

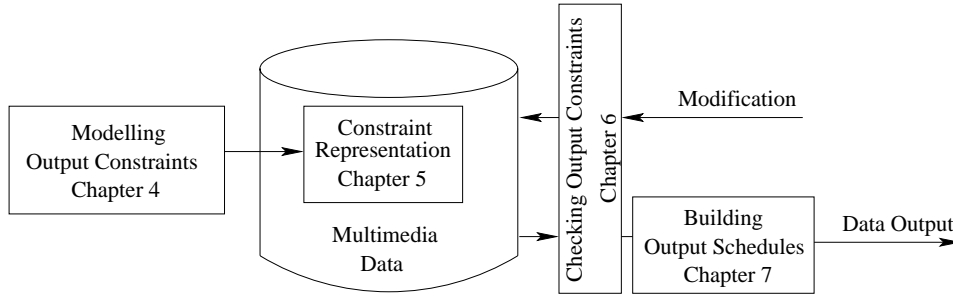


Figure 1.2: Processing of Output Constraints

Figure 1.2 shows several phases we have to consider for managing output constraints. The first topic we must address is modelling output constraints. For modelling output constraints we have to know what kinds of output constraints exist. Thus, we build a classification of output constraints in chapter 4. Furthermore, we introduce a specification language for output constraints.

For checking output constraints efficiently a database internal representation is necessary. We develop this database internal representation of output constraints in chapter 5. Furthermore, a transformation is required that transforms output constraints – specified in the constraints specification language – into the database internal representation. This topic is also considered in chapter 5.

Output constraints must be checked in case of modifications to stored media data and in case of data output. Chapter 6 shows concepts for checking consistency of output constraints. In addition to basic algorithms for checking we propose several optimizations.

For data output we need an output schedule that is built based on output constraints. Chapter 7 deals with the management of output schedules. We show how output schedules are produced and adapted efficiently.

Chapter 8 shows an implementation of output constraints for a practical example. Furthermore, several experiments are done in order to prove the practical benefit of output constraints. Chapter 9 gives a conclusion of this work and shows what can be done in future work.

Chapter 2

Requirements on Output Constraints

This chapter analyses the requirements of some applications which manage multimedia data. We do not a general requirement analysis, rather we analyse the output constraints which are used in these applications.

2.1 Wave Field Synthesis

A new application area for multimedia database systems arise from new developments in the media production area. *Wave Field Synthesis (WFS)* [BV94] is one of these new developments. It allows to produce sound sources on a certain place in a listening room. This *spatial sound* is used in cinemas for giving the visitors the impression of spatial order of sound sources in a movie scene. In order to build the wave field a large number of loudspeakers is needed. They are arranged around the walls of the listening room. Each of these loudspeakers can be controlled separately. To define a wave field, we have to specify the time where loudspeakers are active and the sound which they output.

Up to now only some prototypes of WFS systems are built. The first prototype was built in the cinema of Ilmenau (Germany), a second was built in summer 2004 in Los Angeles near the Hollywood film studios. It is used as exhibition system for film producers. A third system is located in a Virtual-Reality-Laboratory of the University of Surry (United Kingdom). WFS systems can also be used for live entertainment events such as concerts or theater productions.

The technique of wave field synthesis was developed in 1989 in the Laboratory of Acoustical Imaging and Sound Control of the TU Delft. It is based on the wave theory concept of Huygens. This theory says that each point on a wave front serves as a point source of spherical secondary wavelets. It is possible to apply this concept in acoustics by using a large number of small and closely spaced loudspeakers (loudspeaker arrays), as shown in figure 2.1.

The main advantage of WFS systems in comparison to Dolby 5.1 or other sound systems is the ability to position sound sources inside or outside the listening room.

Each listener has its own; natural, spatial correct, and realistic sound impression independent from its position in the room. Inside the listening room sounds can come from any direction. With Dolby 5.1. only few people have the possibility to listen to the sound in the correct spatial way. In contrast to that, people can move freely around within the wave field that is produced in the listening room. At all positions in the room, they have the same natural and spatial correct sound impression. The ‘sweet-spot’ is no longer limited to only few places in the audience. It now fills the complete listening room. Furthermore, sound sources can be moved through the room, and every listener is able to follow the acoustical motion of the sound sources. Regarding the technical setup, more loudspeakers are required than in other sound systems, but no special loudspeaker setup is needed.

WFS systems improve the modelling of sound by supporting an *object-oriented* modelling approach. In contrast to today’s sound reproduction and production systems are based on a *channel oriented* approach. This approach defines the storage and transmission of audio data for a certain loudspeaker setup. Channel oriented means that for each single audio channel the sound must be modelled. In other words, the sound must be modelled for every single loudspeaker. An advantage of this approach is its easy realization and that only a little computational effort during the data output is necessary. Therefore, channel oriented sound systems have a high performance at runtime, but the process of sound production is a very extensive work. It takes a long time and costs much money. Furthermore, produced sounds depend on a specific loudspeaker setup. As mentioned before, for a wave field we need loudspeakers around the walls in the listening room. Thus, the number of loudspeakers is variable. It is easy to see that a channel oriented sound modelling is not suitable for a WFS system.

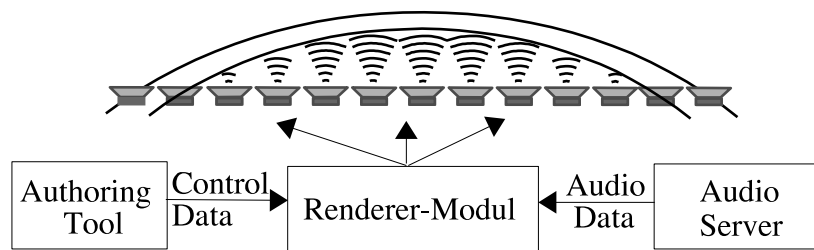


Figure 2.1: WFS-System Architecture

The kinds of data which must be handled in a WFS system and the data flow is shown in figure 2.1. The existing WFS prototypes store the required audio data on an audio server, usually as single audio files.

Beside audio data, control data must be handled as well. These data come from the authoring tool and define the spatial position of sound sources and their temporal order. The current WFS prototypes do not support a relative definition of temporal and spatial output orders, rather the spatial positions and the point in time of output are hard coded. This kind of modelling is very inflexible and an extensive work.

Before we can launch a WFS system its modelling options must become easier. For a comfortable modelling the system must provide possibilities to define the temporal and spatial order of sound sources relatively. Output constraints can be used to do this. As an example, we want to define an overlap of a certain sound source with another sound source, but we do not want to use hard coded time points. So, we need a special *overlap* relationship in our output constraints in order to realize this.

For a real data output the database system must build an output order that fulfills the output constraints. We call this output order an *output schedule*.

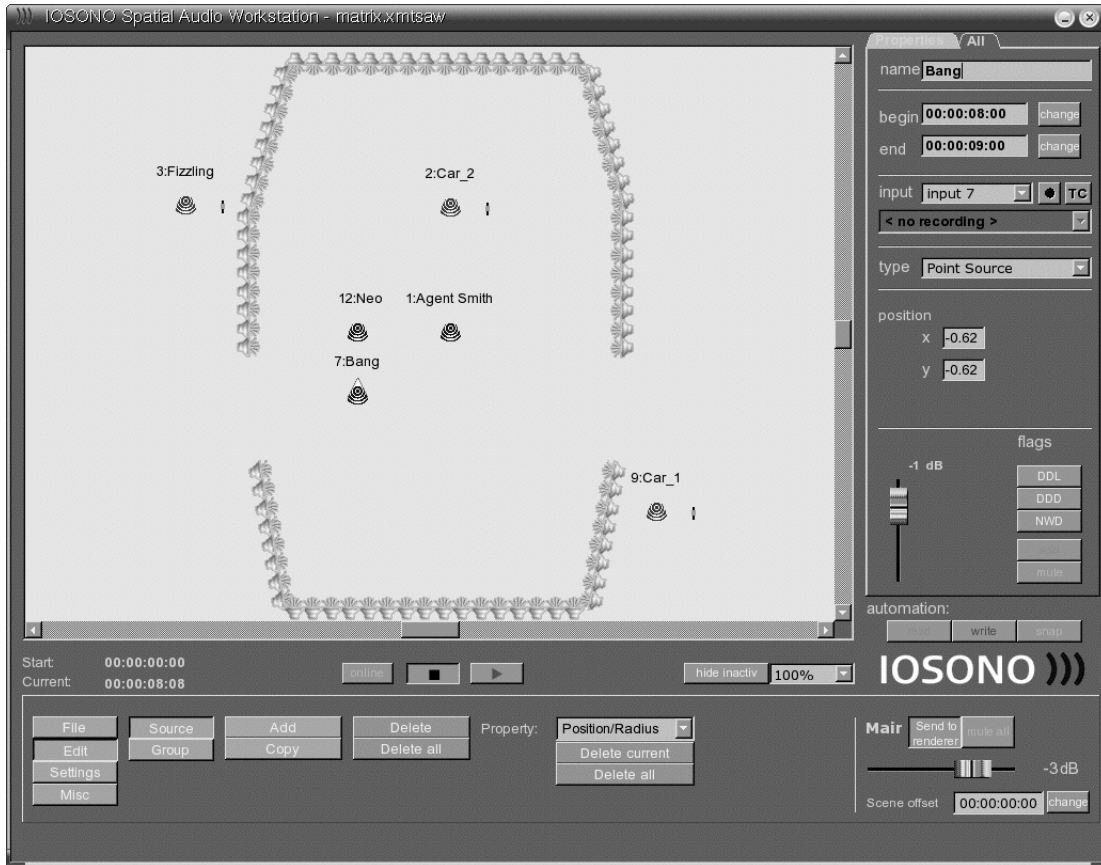


Figure 2.2: Graphical Authoring Tool

Figure 2.2 shows the graphical authoring tool, it is used for modelling the temporal and spatial constraints as well as constraints on properties of sound sources, like a certain loudness for a sound source. At the moment the authoring tool only allows a hard coded definition for spatial positions and temporal orders of sound sources. It stores all these information in an XML file. Thus, the defined temporal and spatial constraints are totally independent from the audio data involved in these constraints.

Beside the modelling of sound sources audio data must be recorded and stored. Usually, this is done without regarding the constraints that arise from the modelling process. So the risk is high to store audio data which are not appropriate to the defined

output constraints. The current WFS prototypes have no consistency concept for this problem. A possible way to solve this problem is using a multimedia database system that manages audio data and output constraints together. Database systems provide possibilities to check data consistency. It is possible to extend this technique in such a way that the consistency between audio data and output constraints can be checked.

The renderer is the core piece of the WFS-System. It builds the actual wave field by using the control data which contain the spatial and temporal descriptions of sound sources as well as the audio data. Both kinds of data are delivered synchronously to the renderer. There are several renderers in a listening room, each one controls some loudspeakers. Inside the renderer the geometry of the wave field is calculated. According to this calculation the loudspeakers are activated by different intensity. Thereby, a listener has the feeling that a sound source is located on a certain place in the listening room. Because of the costly wave field calculation every renderer can manage only eight loudspeakers. For a real cinema we need over a hundred loudspeakers as well as a high amount of renderers to control them.

The actual WFS prototypes have a big drawback, each renderer calculates the geometry of the whole wave field. According to this calculation the renderer can decide whether it must activate its loudspeakers or not. There are several problems that arise from this architecture. First, each renderer needs all control data as well as all audio data. Thereby, much more data are sent than actually needed. The second problem is that each renderer must permanently calculate the wave field. Hence, we need very good hardware for the renderers and a high speed network between the components. If we could discharge the renderer from the complex calculation, we could use much cheaper hardware. This is a very important point if we want to bring the system on the market.

Another challenge is the data organization. The WFS systems use the WAV-format with 24Bit@48kHz for audio data. Therefore, we have a data rate of 48.000 Samples/s · 24 Bit = 1125 KBit/s \approx 1 MBit/s for every sound source. Some movie scenes have 100 sound source. Consequently, we need a data output rate of 100 MBit/s. This is more as most hard disks can provide. We can use several database techniques for optimizing the data organization according to the output constraints. As an example we can use partitioning, indexing or caching strategies.

The actors *author*, *sound producer* and *renderer* are involved in a WFS system as follows:

The Author defines the spatial, temporal and spatiotemporal relationships between audio objects. Composers or sound masters for concerts or movies are possible authors. An authoring tool is used to make authoring comfortably. If the sound for a movie must be defined, often several authors work at the same time on the movie sound, but usually, authors work on different scenes. Thereby, it is not useful to define temporal relations between output objects with absolute time points, rather we want to express temporal relations between audio objects in a relative manner. Modelling

a scene should be very intuitive. This means the relationships used must be easy to understand for the author. It is desired to use Allen-Relations [All83] for modelling temporal relationships between audio objects.

The defined relationships are stored in an XML file. The existing prototypes do not store this information into a database system.

The Sound Producer prepares audio objects. The audio objects must be recorded. Usually, some special software is used to edit the audio data. The sound producer must define constraints on quality parameters for the data output. As an example the minimal sample rate for the data output must be defined.

The Renderer controls the loudspeakers, it produces the actual wave field. In order to improve the current WFS prototypes the renderer should ask a database system for the required audio data. So it does not have to calculate the wave field geometry by its own.

The Output Constraints we need for WFS systems are mainly synchronization constraints. Only constraints on audio data are needed. It must be possible to define temporal, spatial and spatiotemporal output constraints for audio objects inside the database system.

The relationships between audio objects must be defined relatively to each other. For temporal constraints we use Allen-Relationships. An easy way must be found for defining spatial relationships between audio objects. For the wave field synthesis an audio object can be seen as a point in a 2-dimensional space. Hence, we must only model spatial relationships between points. The author also wants to define spatiotemporal relationships. These are constraints which restrict the movement of the output object over a certain time period. As an example an output object be close to another output object for a certain span of time.

We have constraints that arise from physical characteristics. As an example we have defined that the audio object for a bass must be output at most 10 milliseconds after the audio for a guitar. Now we want to scale the room of this scene. As a consequence the spatial distance between bass and guitar increases. We assume that the listener is close to the audio object of the guitar. Because of the larger distance between bass and guitar the bass sound has a longer way and needs more time to get to the listener. Consequently, the listener can hear the bass more then 10 milliseconds after the guitar.

If we define the output constraints relatively between the output objects, we must produce an absolute order between the output objects for the output process. Thus, we have to produce output schedules for the temporal and spatial order of audio objects. The output process must be appropriate to that output schedule. The output constraints must be checked during the data output process. There are different possibilities to react on a constraint violation. A basic reaction would be the abort of the output

process. However, instead of an abort the database system should give a warning to the user of the system. Output constraints must also be checked during the modification of audio data. In this case the database system should not accept audio data which cannot fulfill the output constraints.

Temporal output constraints deal with very small units of time, and a lot of audio data must be output at the same time. The database system must organize the data in such a way that the output schedule can be guaranteed. This means that the database system must use special caching, portioning and indexing strategies.

2.2 Virtual University

E-learning is a popular research field for several sciences. Many educational institutions expect a great benefit from e-learning. Therefore, a lot of e-learning projects have been started. The *Virtual University of Bavaria* is one of these projects. The intention of this project is to build a platform for multimedia course offerings. That means videos, slides, and audios from lectures are stored. The students can access these data through the internet. This frees them from fixed timetables and they can take their lessons on any place they want. Furthermore, the students can repeat the lessons and they can choose the specific information they need. The way of learning can be determined by the students themselves, so they can learn more efficiently. At the moment it can be regarded as an addition to normal lectures, but a positive impact from that extra offer can be recognized. The final goal is that the Virtual University of Bavaria becomes equal to a normal university.

The virtual university needs both alphanumeric and media data. The data management can be done either by a multimedia database system or by a file server. The advantage of the database system in comparison to the file server is that it provides more functionality for searching and transactions. Therefore, a multimedia database system is more suitable for this application.

In this application media data and output constraints are stored at different places. Media data are stored inside the database system, in contrast to this output constraints are hard coded in HTML pages or in SMIL scripts. Usually, these scripts are not stored in the database. Media data can be changed regardless of output constraints. Thus, inconsistency between stored (multi)media data and output constraints can easily occur. As an example a video and an audio shall be output equally, but the length of the video is changed in such a way that it afterwards differs from the audio length. Thus, an equal output is not possible. This error can be recognized only during the data output process.

Problems can also occur if availability of the required resources cannot be guaranteed during the data output. As an example a minimum bandwidth should be defined for a video output, output constraints can be used for this purpose. The database system must observe the bandwidth during the output and must react if the output constraint is violated.

The Virtual University has two actors *modeler* and *consumer*, which interact with the system as follows:

The Modeler designs the multimedia documents for the lectures. The design process includes the layout definition (e.g. colors, fonts) and the definition of spatial and temporal relationships between media data.

As an example a professor wants to integrate teaching material into the system. Two steps are necessary for that. First the media data (e.g. videos) must be inserted into the system. In a second step the modeler has to define a multimedia document based on the single media data. The multimedia document describes constraints between the single media data as well as their presentation.

The Consumer wants to get information about lectures. Usually, the data consumer poses queries to the database system in order to get the required information. Several kinds of queries can be distinguished, the easiest group of queries requests only alphanumeric data. Another group asks only for single media data. As an example, a user can ask only for the audio of a lecture. The most complex queries ask for whole multimedia documents. It is often desired that the user can interact with these documents. It is important to note that the number of possible queries is very restricted. This means only predefined queries are possible, the user cannot define ad-hoc queries.

Usually, the system is used via an internet browser. To keep the system simple we do not install extra client software. So, the server must produce a HTML or SMIL document as query result. The server must be able to output the media data in the required temporal order.

Students are consumers of the Virtual University. They send requests to the database system and want to get information about lectures. Usually, they are looking for a specific topic of a lecture. Often, they do not want to see the entire lecture information, rather they only need a small part of the stored information. We can use the stored metadata for searching, but there are also some query languages [LÖSO97] for building semantic queries directly on media data (e.g. videos). The database system is responsible for the correct data output of the multimedia data. This means that the database system must maintain all output constraints which are defined on the required output objects.

The Output Constraints we need for this applications are mainly synchronization constraints. The modeler defines temporal and spatial constraints between different kinds of media data. Usually, the temporal and spatial order of media data is hard coded with absolute numbers. We can do this because the media data belong to only one multimedia document. This kind of modelling is analog to modelling the output constraints by using SMIL. An advantage that arises from this is that output constraints can automatically be generated from SMIL scripts.

It is possible that output constraints can be violated by data inputs or updates. As an example if a video and an audio must be output equally and the length of one of them has been changed, an equal output is no more possible. If we allowed those changes, the output consistency could not be achieved. This means that output constraints are violated.

The violation of output constraints during the output process is also possible. As an example if we must handle many output processes at the same time, the read rate from hard disk can become a bottleneck. Thus, the database system must check the output constraints during the output process. If a violation is detected, we have several possibilities to react on it. The hardest solution is to abort the complete data output process. A more adequate way is to continue the data output and try to tune the system online.

2.3 Hospital Database Application

In hospitals we have the situation that many specialists (e.g. internist, orthopaedist) work together, but often each one maintains its own patient records. It is a simple fact that the same patient data (e.g. x-ray image) are needed by several specialists for diagnosis. It is quite evident that this often leads to a repetition of examinations. To solve this problem a hospital can establish a central database system which manages all data. This system must be able to handle many data output processes at the same time. Usually, each output process deals with large amounts of data.

In the medical science a lot of examination methods are used which produce images and videos. Therefore, lots of x-ray images, ultra sound videos, and videos that originate from computed tomography must be handled in a hospital. Furthermore, alphanumeric patient data like patient name or insurance policy number must also be managed. The challenge is to handle both kinds of data efficiently in one database system. If we have a central data source for all applications, we can realize *electronic patient files*. Such a file contains all information about a patient, particularly it must include all media data. With such an electronic patient file we can avoid the needless repetition of examinations.

Each computer application that is used by a specialist must be connected to this central data source. The specialist often needs media data or multimedia documents for the diagnosis. As an example a multimedia document can consist of two ultra sound videos. One shows the situation before a surgery the other has been made after the surgery. Both videos should be output equally in order to determine the effect of the surgery. Today it is usual that multimedia documents are built by applications. Usually, the database system only provides the media data to the application. All constraints which are necessary for building a multimedia document are part of the application logic. The well-known problems of this approach are redundant implementations of output constraints in each application and possibly unintentional modifications on media data which make the output constraints unrealizable.

From an abstract viewpoint the actors *data producer*, *modeler*, and *data user* interact with the hospital database application as follows:

The Data Producer is the medical system which generates the data. A system is the combination of a device (e.g. fluoroscope) and the control software for this device. Normally, a system only generates single media data. As an example the fluoroscope only produces images.

It is important to note that the data quality is a crucial factor for medical data. The resolution of an x-ray image is an example for this. If the resolution is too low, some details in the image can be lost. This means the data become semantically incorrect. So, it is useful if the producer of data could define constraints on output parameters in order to maintain the semantics of the data. As an example the x-ray application must be able to define a minimal resolution for the x-ray images.

The Modeler defines the required multimedia documents. Usually, special software systems assist doctors to build diagnoses. Different specialists often need the same multimedia document. The aforementioned multimedia document which contains two ultra sound videos – one before and one after a surgery – is of interest for the internist and for the surgeon. If some applications use the same multimedia document it is useful to model it outside these applications.

The modeler designs the multimedia documents by defining spatial and temporal relationships between the media data and constraints on output parameters.

The Data User asks for (multi)media data. All kinds of applications which need data are data users. It is very important that the required data have the defined data quality. The database system must guarantee this quality during the output process. Furthermore, the database system must guarantee consistency between stored media data and output constraints in case of modifications.

The Output Constraints needed for this application are synchronization constraints and constraints for quality parameters. In contrast to the Virtual University of section 2.2 temporal and spatial relationships are modelled relatively between media data. As an example we could define that two ultra sound videos should be output equally, regardless of the exact start and stop time of these videos.

Because of the relative definition of temporal and spatial constraints an absolute temporal and spatial output order must be built for the output process. The database system has to build this output schedule by using the output constraints. Furthermore, the data organization must be suitable to the output schedule.

Output constraints for temporal or spatial synchronizations can be violated by manipulations of media data which are involved in this constraints. The database system has to prevent this kind of manipulations. Furthermore, constraints on output parameters can be violated during the output process. The database system must detect these

violations and an appropriate reaction must be executed. There is a wide spectrum of possible reactions that includes sending a message to the user as well as aborting the output process.

2.4 Conclusion

	Virtual University	Hospital Database	WFS-System
Kind of multimedia data	Video, Audio, Full Text, Image	Video, Audio, Image	Audio
Output Constraints	Synchronization, Quality Constraint	Synchronization, Quality Constraint	Synchronization
Definition of Synchronization	hard coded time points	relatively between output objects	relatively between output objects
Time-point of Constraint check	during data input, during data output	during data input, during data output	during data input, during data output
Reaction on violation during data input	transaction abort	transaction abort	transaction abort
Reaction on violation during data output	self-tuning of the data output process	output abort	warning message to the system administrator

Table 2.1: Requirements for Output Constraints

Table 2.1 shows the requirements on output constraints which arise from different applications. It is important to note that we do not support output constraints which take user interactions into account. Examples of user interactions for an audio are *play* and *stop*. This group of constraints has been considered in [VB97].

It is a requirement of the considered applications that it must be possible to define output constraints on any kind of media data. Furthermore, we can identify synchronization constraints and constraints on output parameters as the main constraint groups, that we need.

How synchronization constraints must be defined depends on the application. As an example in the virtual university application (section 2.2) we define temporal synchronization constraints between output objects by using absolute time points. In contrast to this in a WFS system we must model those constraints relatively between the output objects. Therefore, it is necessary for the data output to produce an absolute output order. This means that the database system must generate an output schedule before outputting data.

All applications need a constraint check during the input or modification of media data and during the data output. If output constraints are violated by data modification

operations, we can handle this violation like a traditional integrity constraint violation, that is why we try to use integrity constraints in this point.

Chapter 3

Related Work

Multimedia database systems are a basis of this work, we want to improve these systems by integrate output constraints into them. Therefore, we must consider the latest developments in this field of research. From the requirements analysis (chapter 2) we have seen that output constraints come very close to the integrity concept of database systems. We can even say that our output constraints are an enhancement of the traditional integrity concept. That is the reason why we must consider the state of the art for integrity constraints. Presentation of media data is well-known in the field of multimedia documents. Several presentation models have been developed there. We have to consider these models and determine their usability for our output constraints. Synchronization models and other approaches are proposed for managing consistency of multimedia data. We have to consider these works because they are a basis for our output constraints.

3.1 Data Models and Architecture of Multimedia Database Systems

There is no general definition of multimedia database systems. Thus, we begin with considering requirements on multimedia database systems that are important for this work. Afterwards, we give a short overview of data models and show a possible architecture of a multimedia database system. At the end of this section we consider some multimedia database systems that are related to this work.

3.1.1 Requirements on a Multimedia Database System

The term *multimedia database system* means different things to different people. This is not surprising because *multimedia* can imply many different things. In this section we present a perspective on and a definition of multimedia databases, which is used in this work.

A *multimedia system* [Mar92] is a computer based integration of information objects from different media types (text, audio, image, video). The integration covers the points: data modelling, data storage, data presentation and temporal synchronization.

From this definition it is easy to see that a multimedia database system is only one part of a multimedia system. The multimedia database system is that part of the multimedia system which stores the multimedia data.

A *multimedia database system* [Spe98] has all characteristics of a traditional database system. In contrast to this, it manages not only alphanumeric data but also a high amount of media data on several storage media. These storage media includes main memory, hard disk and optical storage media. Data retrieval, data access and data manipulation must be efficient for alphanumeric and media data [KB96]. Furthermore, it is assumed that all media data are digital and stored on a computer.

Several differences to traditional database systems arise from the fact that a multimedia database system must manage time dependent data as well as very huge single data objects [KA97]. In the following we only mention a few of them, which have influence on defining or checking output constraints:

Extendible data types are important for media data. Traditional database systems only support a few data types. This leads to the fact that media data are usually stored as binary large objects (BLOB). All kinds of media data (text, audio, image, video) are stored using this data type. However, different media data must be handled in different ways during data output. If a BLOB covers all these data types, it is difficult to handle each kind of media data individually.

Access on data slides is very important for media data. As an example if we want to output only one scene of a video, a direct access to this scene data is needed. Thus, a multimedia database system must provide techniques for efficient access on parts of large media objects.

Index structures for media objects are very different from traditional index structures. In traditional database systems an index is built on one or a few attributes. Media data have a lot of characteristic features which must be considered during data retrieval. Thus, a multimedia database system must support high dimensional index structures. Furthermore, special temporal and spatial index structures are necessary.

Device and format independency is a well-known feature of traditional database systems. Thus, multimedia database systems must support this feature for media data as well. This means that the internal storage format for media data is independent from the data format which is used for the data input or data output. Therefore, the database system must be able to transform media data from the internal storage format into several output formats. This is a big challenge for time dependent media data, e.g. video. For these data types the format transformation must be accomplished in real time [MR97].

The requirements on a multimedia database system mentioned before are well-known from literature. To the best of our knowledge there is no requirement concerning the integrity of media data. We want to put the requirement of *output consistency* on the above list.

Supporting multimedia documents is another requirement we have to make on multimedia database systems. These documents consist of several media objects and relationships between them. Media objects as well as relationships must be managed by the multimedia database system.

All the aforementioned requirements on a multimedia database system can be summarized in the following working definition:

A multimedia database system must be able to manage alphanumeric data as well as a high amount of media data. Storing of media data must be device and format independent. Efficient storage structures for media data and retrieval methods must be supported as well as an adequate multimedia query language. In order to build complex multimedia documents constraints between media data must be provided. Furthermore, output consistency must be guaranteed by the multimedia database system. The multimedia database system should also provide a graphical user interface and an application interface.

3.1.2 Data Models for Multimedia Database Systems

A data model [AHV95, KBL05] consists of a set of concepts and languages for describing these concepts.

Conceptual and external schemas. A schema specifies the structure of the data that are stored in the database. Schemas are described by using a data definition language (DDL).

Constraints. A constraint specifies a condition that the data items in the database must satisfy. A constraint specification language is usually part of the DDL.

Operations on data. These operations give us a high-level data abstraction. A data manipulation language (DML) is used for describing these operations.

The Relational Data Model [Cod70] is still the most popular data model in practice. Furthermore, it is the elementary data model. The structural part of the relational model supports relation schemas, relations, attributes, domains and integrity constraints. The operational part supports operations on these relations, like selection, projection, join, and set operations. SQL is the common language which can be used as DDL, DML and constraint language.

Relational database system provide very efficient implementations of operations, like joins. Therefore, they are very fast in link tracing [Spe97], which is an important

operation for hypermedia documents. On the other side, the relational model does not support structured data types. Usually, relational database systems support only data types like BLOB.

The Object-Oriented Data Model is close to concepts coming from object-oriented programming languages. The structural concepts include a type system, classes, objects, object identities and methods [ABD⁺89]. An object definition language (ODL) is used for defining structural elements. The object-oriented data model supports generic operations like retrieval of objects in classes and specific operations implemented by methods.

This data model is suitable for a multimedia database system. Concepts like structured types, inheritance and methods are very useful for handling multimedia data. In practice powerful object-oriented database systems are not on the market.

The Object-Relational Data Model [SBM99] is an extension of the relational data model. It has features like structured types for attributes, methods and identifiers for tuples. The new SQL:1999 standard takes this extensions into account.

Object-relational database systems can handle media data very well. Existing systems like Oracle and DB2 have their own extensions for managing media data. For the sake of simplicity we mainly use the pure relational data model in this work. However, for some examples we use object-relational extensions, like structured types, for a better understanding.

3.1.3 Architecture of Multimedia Database Systems

The contribution of this work is to integrate output constraints into a multimedia database system. Thus, the architecture of existing multimedia database systems is of interest. Figure 3.1 shows an architecture of a multimedia database system which was used in the projects MultiMap [Spe97] and OMNIS [SB00]. The component *Previewer* is usually a stand-alone software and not part of a multimedia database system. The multimedia database system consists of a *Multimedia-Engine* and a *Multimedia-Interface*.

The *Multimedia-Engine* has all components which are known of traditional database architectures. Of course, special requirements which arise from the characteristics of multimedia data must be considered. The index structures exemplify this. Traditional database systems use some variants of B-Trees and Hash techniques for managing indices. These techniques work well for indices up to ten dimensions, but for media data (e.g. images) we must handle indices on high dimensional feature vectors. Thus, the database system has to support special index techniques such as VA-Files [WBS00].

The *Multimedia-Interface* contains components which realize a pre-processing of the database query and a post-processing of the query result. Usually, a special multimedia query language is used for a multimedia database system [LÖSO97]. The

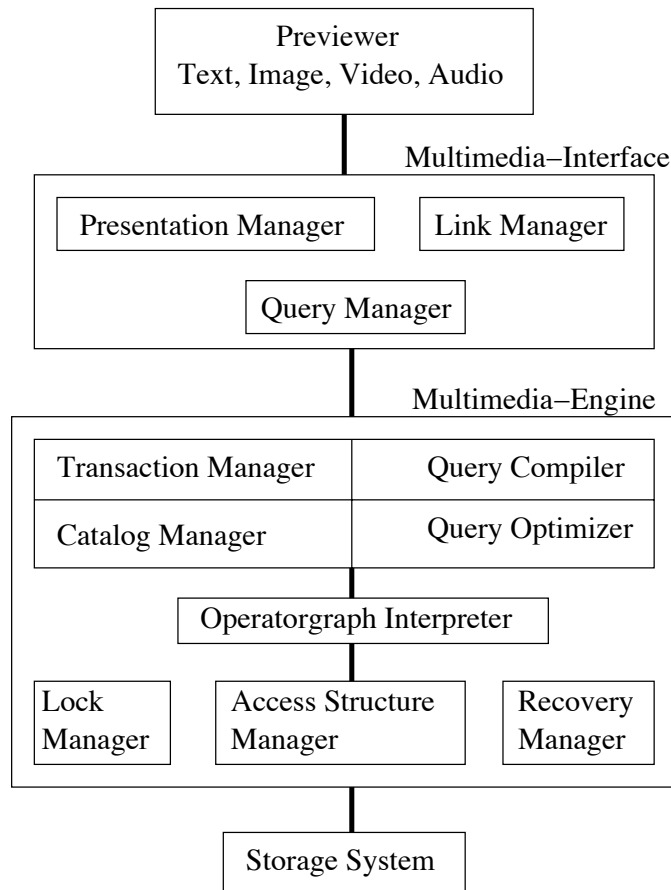


Figure 3.1: Architecture of a Multimedia Database System [Spe98]

Query Manager transforms a query defined in a multimedia query language into SQL-Statements, which will be sent to the Multimedia-Engine. The user of a multimedia database system often wants to get a multimedia document that consists of several single media data. Therefore, the original database query also includes information about the links between media data and information about presentation parameters such as temporal and spatial relationships between the data. The *Link Manager* builds the static links between the media data and the *Presentation Manager* generates the temporal and spatial order according to the requirements that arise from the original database query.

The architecture of figure 3.1 only considers the output constraints which are part of the database query. In other words, only the database user can define how the stored multimedia data should be presented. The original data producer cannot define the way of data output. If the user modifies the data output in any way, a semantical falsification is possible, even in an unintentional manner. The described architecture does not check the output constraints during the output process, thus defined reactions on output constraint violations are not possible.

If we want to integrate output constraints into a multimedia database system, we have to modify its architecture. An additional component must be integrated into the Multimedia Interface. This component must check the output consistency during the data output as well as in case of data modifications.

3.1.4 Existing Approaches for Multimedia Database Systems

There is a large number of existing multimedia database systems, both commercial and research systems. In this section we only mention those systems which are either strongly related to this work or important commercial systems.

Realtime Data Output for Media Database Systems has been investigated in the projects RETAVIC [SMMW04], KANGAROO [MR97], and MOSS [KMMW93]. The challenge was to bring format independency for media objects and realtime together in one system. This means that the user can define the output format of media objects in a database query, the database system has to transform the internal storage format into the required output format. For special media objects, such as videos, hard temporal limitations exist for this transformation.

It is possible to define specific quality parameters for the transformation, but to the best of our knowledge there is no possibility for the producer of media data to define constraints for quality parameters.

Object-Oriented Multimedia Database Systems have been proposed in [OIS⁺97, LÖSO97, OIÖ98, Adi96, Vaz96]. The idea behind this approaches is to use object-oriented techniques for modelling multimedia objects. These concepts, like classes, methods, and inheritance, are very suitable for media data. Therefore, some researchers postulate object-oriented database systems as imperative basis for a multimedia database system [WK87].

Multimedia object query languages are proposed in [HK96, LÖSO97]. A multimedia object query language of particular interest is MOQL [LÖSO97]. The contribution of MOQL is to integrate spatial and temporal predicates into an object query language. This means that the user can define spatial and temporal predicates for salient objects in database queries. Furthermore, a new clause '*presentation layout*' is added as direct extension to OQL. The layout consists of the following three components:

- The *spatial layout* specifies spatial relationships of the presentation, such as size and location of windows (salient objects).
- The *temporal layout* specifies the temporal relationships of the presentation, such as the temporal order of media objects and how long the presentation should last.
- The *scenario layout* allows a user to specify both spatial and temporal layout using other presentation models or languages.

The following example originates from [LÖSO97] and shows the power of MOQL. The query should find all the image and video pairs from the tables *Images* and *Videos* such that the video contains all the cars in the image, show the image in a window with the corner points $((0,0),(300,400))$ and the video in a window at $((301,401),(500,700))$, and start the video 10 seconds after displaying the image; display the image for 20 seconds, but play the video for 30 minutes:

```

select    m,v
from      Images m, Videos v
where     for all c in (select r from Cars r where m contains r)
           v contains c
present   atWindow(m,(0,0),(300,400)) and atWindow(v,(301,401),(500,700)) and
           play(v,10,normal,30*60) parStart display(m,0,20)

```

Some special operators for synchronization are integrated into that query language. An example is the operator *parStart*, it starts two media objects simultaneously.

It is easy to see that this powerful query language can be used for defining any kind of presentation consisting of stored media objects. The output relationships defined by this query language can be seen as user defined output constraints. Thus, implementation concepts for output constraints can be used for realizing this query language.

Commercial Database Systems, such as ORACLE and DB2 as well as free database implementations, such as PostgreSQL, support several media types. Usually, the multimedia extensions of those systems provide special data types for media data, special integrity constraints or output constraints are not supported.

Several aspects considering the output of multimedia objects have been investigated, but to the best of our knowledge up to now there is no project and no approach that considers output constraints.

3.2 Integrity Constraints

The complexity of modern database applications requires powerful facilities for controlling semantic correctness of the data in the database. In traditional database systems we use *integrity constraints* to ensure the correctness of the stored data. The research field of integrity constraints is as old as the database research. There are a number of research topics in this field. We can classify them by the following three categories [GA93]:

Constraint types and semantics addresses various types of integrity constraints in the field of traditional database systems.

Constraint specification deals with various approaches for specifying integrity constraints.

Constraint enforcement and preprocessing attends to various approaches to enforce constraints and the execution of violation response actions. Preprocessing means for example the verification of new constraints, i.e. checking if new constraints are correct and meaningful with respect to a number of criteria.

In the following sections each of these topics is discussed.

3.2.1 Types and Semantics of Integrity Constraints

Static- and dynamic integrity constraints are the most noted classes of integrity constraints. Table 3.1 gives an overview of these classes and the corresponding subclasses.

Static	Attribute	Domain
		Nonnull
	Tuple	Attribute comparison
	Relation	Uniqueness (key)
		Functional dependency
		Aggregate
		Transitive closure
	Database	Referential integrity
		Interrelation aggregate
Dynamic	Attribute	Domain
	Tuple	Attribute comparison
	Relation	Aggregate
	Database	Interrelation aggregate

Table 3.1: Taxonomy of Integrity Constraints [GA93]

We want to consider constraint types that are of special interest for multimedia data. As aforementioned a group of output constraints restricts output parameters, more precisely it restricts the values of output parameters during the data output. Thus, it is close to ‘domain’ and ‘nonnull’ integrity constraints. Referential integrity constraints are interesting because hyperlinks in multimedia documents are often implemented using referential integrity constraints [Oom99].

For checking dynamic integrity constraints different database states must be considered. Some groups of output constraints deal with output parameters that changing over time, like the frame rate of a video. Thus, several time points (states) during the output process are needed for checking these output constraints. We will consider dynamic integrity constraints because they are close to several types of output constraints.

Temporal constraints can be seen as a special group of transition constraints. They originate from the field of temporal database systems and they work on the his-

tory of a database. We have to make a distinction between these temporal integrity constraints and those temporal output constraints for synchronization of media data.

Data privacy becomes more important for database management systems. Data privacy means that data producers have control over who is allowed to see their information and for what purpose. In practice the database system supports rules for restricting the data output. This concept is called *limited disclosure* [LAE⁺04], it is applied in statistic and hippocratic databases. Because limited disclosure deals with data output restrictions, we must consider it in this work.

Domain and Nonnull Constraints

Domain constraints restrict values of attributes. This means that an attribute can adopt only those values that are permitted by the domain constraints. Usually, domain constraints specify a certain range of values for attributes. Sometimes the range of an attribute is defined by an enumeration of possible values. In the relational model the enumerated values can be stored in a separate relation. So, it is possible to implement this domain constraint as referential integrity constraint. As it is shown in [dB89], ‘variable enumerated domain constraints’ can be defined in this way.

Nonnull constraints also restrict the domain of an attribute. Therefore, they can be seen as a special kind of domain constraints. They are considered in [Gol81] more precisely.

Referential Integrity Constraints

The concept of referential integrity is a central issue in the relational and object-relational data model. We can use it for specifying semantic links between various relations in a database. Therefore, this type of constraints has received some attention.

The original idea of referential integrity is described in [Cod79]. An enhancement of referential integrity is given in [Dat81]. An important extension is that foreign keys can contain *null* values. On the other side this can lead to various kinds of problematic situations. As an example it is hard to describe the semantics of referential integrity constraints with foreign key values that are partly null.

Dynamic Constraints

Mostly, in literature we can find transitions or dynamic constraints as ‘single-step’ transition constraints [eS97]. This means that those constraints are evaluated on a pair of pre-transaction and post-transaction states of a database. Usually, these constraints are tuple constraints, i.e. they relate old and new attribute values in a single tuple.

Temporal Constraints

Sometimes it is desired to describe dynamic constraints that are not limited to single-step transitions. Thus, *temporal constraints* take the whole database history into account. Special temporal qualifiers, like *always* and *sometime* [ELG84], are used for defining this constraints. *Always* can be used to specify that something must be fulfilled for the entire database history. *Sometime* is used for constraints that must sometimes be fulfilled in the database history.

We can see from [ELG84] that the enforcement of general temporal constraints is very inefficient. Restricted temporal constraints are discussed in [Cho92]. There a method is described for the enforcement of temporal constraints taking only past database states into account. This method makes use of redundant information to be able to enforce temporal constraints without storing the entire history of a database. We can see a similar idea in [PTJ01]. There a graph is used for modelling temporal constraints. Thus, it is not necessary to store the database history for a constraint check.

Output Constraints in Statistic Database Systems

For security reasons private data in statistic and hippocratical databases need a special protection. Therefore, rules are introduced which restrict the data output. The producer of data can use these rules to protect its information. In practice the result set is restricted by the rules. As an example very small result sets are not allowed. Generally, these rules can be seen as output constraints, because they restrict the data output. However, they do not guarantee the semantics of the data output like the output constraints proposed in this work.

To the best of our knowledge up to now database systems do not support output constraints which guarantee data semantics during the output process.

3.2.2 Specification of Integrity Constraints

We have discussed some meaningful integrity constraints in section 3.2.1. This section deals with alternative kinds of constraint specification. We can consider the following kinds of specification:

declarative: The integrity constraints are defined by using a data definition language.

operational: The integrity constraints are defined by using a programming language. This can be done inside a database application or in stored procedures as part of the database system.

rule based: We can use rules, usually Event-Condition-Action-Rules, for defining integrity constraints.

Declarative Specification of Integrity Constraints

The declarative approach of specification is based on relational algebra. A common way to express a constraint is to specify a relational expression that calculates the tuples in the database that do not satisfy the constraint [Gre93]. Some approaches add special-purpose constructs to the relational algebra in order to specify constraints. As an example in the PRISMA [AvBF⁺92, WGAK89] project all constraints are defined in a first-order logic. These constraints are translated into an extension of the relational algebra, called XRA [Gre92]. Therefore, they are directly executable by a special system component.

Usually, the SQL language is used for constraint specification [ABC⁺76, CW90]. Today's database systems support the integrity constraints shown in table 3.1 as declarative integrity constraints [TG01].

Operational Definition of Integrity Constraints

We can use stored procedures or methods in an object-relational database system for defining integrity constraints. So, checking integrity constraints can be encapsulated into class methods or stored procedures. This way of integrity control has the advantage that a programming language, like Java, can be used for implementing integrity control. On the other side there are a number of disadvantages. The implementation of complex integrity constraints is not trivial. We have to consider that possibly several stored procedures affect others. Another point is that the integrity constraints are defined inside the stored procedure. This means that the database system cannot use those integrity constraints for query optimization. Furthermore, the modelling of output constraints inside stored procedures leads to many procedures with very complex implementations.

Rule-Based Definition of Integrity Constraints

Event-Condition-Action-Rules (ECA-Rules) can be used for defining and enforcing integrity constraints. ECA-Rules are also known as active rules, since with this kind of rules the database system can react actively on defined situations. Usually, a certain action is executed when a defined situation occurs. The components of an ECA-Rule are defined as follows [CW90]:

Event: This component describes a specific situation (event) which is the trigger event. We can distinguish *primitive events*, as an example the start or end of an operation, and *composed events* that are compositions of primitive events.

Condition: This part of an ECA-Rule specifies a condition that must be fulfilled for the execution of the defined action. It is possible to use a database query as condition. In this case the condition is fulfilled if the result set is not empty.

Action: This part defines the action that must be executed when the defined event occurred and the condition is fulfilled. Usually, this is a sequence of method calls or the transaction abort.

The following general SQL notation for integrity rules was originally proposed in [ABC⁺76, Dat83]:

```
assert < Constraints Name >
[immediate | deferred] [ on < Operation > ]
[for < Relation > ]: < Condition-Clause >
[else ( < SQL Statements > )]
```

The *event* defined in the ECA-Rule can be mapped to *operation*, the *condition* can be mapped to *condition-clause*, and *action* can be mapped to *SQL statements*.

Triggers are a special group of ECA-Rules that are well-known from relational database systems. Usually, triggers can only detect database events, i.e. updates or deletes. In real database systems we have to specify a certain coupling mode for a trigger. This means that we must define whether a trigger is executed immediately after a triggering event or at the end of the transaction.

The rule based definition of output constraints has some consequences that must be considered. ECA-Rules can activate each other, this can lead to a not terminated situation during the execution of ECA-Rules. Some work has been done dealing with this termination problem [BW00]. Another problem occurs when a set of different ECA-Rules must be executed on a certain event. So, we have to define an execution order for the ECA-Rules. Maintaining ECA-Rules is a general problem and a sophisticated task because it is difficult to analyse the netting of ECA-Rules.

The advantage of the rule based approach is that it gives the database designer the chance of building very smart triggers and of enhancing the efficiency of the constraint enforcement. Of course, in order to do so the database designer needs detailed knowledge about the database.

3.2.3 Checking of Integrity Constraints

The process of constraint checking contains the following four parts [GA93]:

Constraint verification checks if constraints are syntactically and semantically valid [Bro78].

Constraint translation translates constraints from the constraint definition language into a representation which is more suitable for constraint enforcement [GA91].

Constraint optimization transforms constraints in such a way that the execution costs of constraints enforcement are minimal. The execution cost of constraint enforcement is one of the major problems in constraint handling. Thus, constraint

optimization is very important. This topic is very similar to query optimization [Tür99, CP84].

Constraint enforcement deals with constraint prevention and detection. Furthermore, different kinds of system responses to constraint violation are considered in [GL97].

For checking output constraints these four steps are necessary. Since we can use some ideas for managing output constraints, we want to consider constraint verification and constraint enforcement in detail.

Constraint Verification

The following points must be verified when a new integrity constraint is defined:

- Syntactic correctness must be checked with respect to the syntax rules for specifying constraints. After the syntax check a semantic check must be done. As an example it must be checked if all used relations and attributes exist, and if all comparisons are made only between compatible operands.
- We have to consider the relationship between the set of already defined constraints and the new constraint. It must be checked if the new constraint is not contradictory to already existing constraints. This can be done by checking the consistency of the constraint set. Valid database states or transitions can only exist if the set of constraints is consistent.
- It must be checked if the new constraint is implied by already existing constraints. If this is the case, the new constraint is superfluous. A redundant set of constraints is undesired for efficiency reasons.
- In case of using triggers it must be guaranteed that the violation response action does not trigger an infinite process of compensating actions due to the interaction with other constraints [CW90].

Constraint verification is also necessary for output constraints. All the points mentioned before must be taken into account for handling output constraints.

Constraint Enforcement

Violation prevention and *violation detection* are two general techniques for enforcing integrity constraints [GV89].

Violation detection is used if integrity constraints are enforced after the updates of a transaction is applied to the database. In case of violations the transaction is aborted and the changes are made undone.

Violation prevention is used if integrity constraints are enforced before the updates of a transaction are actually applied to the database. Usually, the transaction is aborted

when the constraints are violated. An undo operation is not necessary because the database state is not changed. To realize violation prevention we can use *transaction analysis*. This technique tries to analyse update transaction before they are post on the database system. A transaction is *safe* if it cannot violate the integrity of any database state.

Another technique for violation prevention is *query modification*. In this approach updates on the database are modified in such a way that they cannot violate the integrity of the database [Sto75]. After these modifications the updates can be executed without any further checks. However, query modification is not applicable to all constraint types [SV84].

3.3 Presentation of Media Data

Initially, the management of multimedia data on a computer was not done by a multimedia database system. Actually, the user had the wish to use images and other media objects in internet presentations, therefore, a presentation model was needed and a certain data quality must be guaranteed. These are the topics of this section.

3.3.1 Quality of Service

Quality of service (QoS) is generally important for all kinds of multimedia data. The notation of quality of service emerged from communications to describe certain technical characteristic data of transmission. QoS parameters for data transmission mostly apply on lower protocol layers and are not meant to be directly observable or verifiable by an application. As time-dependent data became prevalent in multimedia-applications the entire system need to participate in providing the guaranteed performance level. Therefore, QoS parameters can now be found on communication level, on operating system level and on application level (e.g. in a multimedia database system) [BEH01, BEH03]. If we take the output of a stored video as an example we can find the quality parameters *throughput* on transmission level, *Signal-to-Noise Ratio* [Ohm04] on operating system level and *delay* on application level (database system).

If we wanted to give a definition for QoS, we could only do this in a very general manner. We use the definition of [VKvBG95]: ‘*Quality of service represents the set of those quantitative and qualitative characteristics of a multimedia system necessary to achieve the required functionality of an application.*’

Functionality includes both presentation of multimedia data to the user and general user requirements. In other words, we have quality parameters that arise from technical environment like a network’s bandwidth and other quality parameters directly given by the user, e.g. a specific image resolution. The QoS of a given system is expressed by a set of parameter-value pairs. These parameters are from different system levels (transmission, operating system and application level), some parameters might not be mutually independent. This fact must be taken into account when a certain level of

quality is supposed to be guaranteed. Therefore, parameters are often not considered separately, rather a parameter space is considered where each parameter is assigned to a dimension of that space [SWM95]. A special function takes all quality parameters and computes a value of the parameter space, best quality is given when this value reaches an optimum.

When dealing with quality parameters on different system levels, a serious problem is to map quality parameters from the higher levels to adequate parameter-value pairs of lower system levels. As an example, parameters of video quality on the application level (e.g. delay) must be mapped into parameters of the transmission level like *throughput*. Some important work in this field has been done by Goebel and Plagmann [GP99, EGPJ02].

In this work we assume that a parameter mapping is given and values can be detected for all required quality parameters. There are several ways of specifying quality parameters. Beside the parameter-value pairs mentioned before a specification using difference constraints is proposed in [CRS98]. To simplify matters in this work we use parameter-value pairs for defining output constraints on quality parameters. These pairs can be defined much more intuitively than difference constraints.

3.3.2 Presentation Models for Multimedia Data

Several presentation models and languages have been developed, HTML and SMIL are common examples. So, it became possible to build complex multimedia documents out of single media data which are stored as files in the file system. Usually, a presentation model defines several ways for temporal and spatial synchronization of media data.

There is a high number of presentation models, a good overview to that topic is given in [MW03, BF98]. This section only refers standards, like ODA [Hor85], generic reference models, like Dexter [HS90], and SMIL [Bul01] as a presentation language with great practical importance.

SMIL

The ‘Synchronized Multimedia Integration Language’ is a multimedia extension of XML [Bul01]. SMIL 2.0 exists since August 2001 and has great practical relevance since it is supported by many multimedia and internet browsers.

SMIL supports temporal and spatial synchronization for several media data. Usually, media data are stored in appropriate formats within conventional files. To address media data within a SMIL-Script a path expression is used. There is no possibility for a SMIL-Script to recognize a change of that path or of the media file. A mismatch between the stored media data and the defined synchronization can occur easily. Therefore, the risk of errors is very high. Techniques for maintaining consistency between media data and SMIL-Scripts are not considered by the SMIL standard.

Open Document Architecture (ODA)

The reason for developing ODA [Hor85, Krö88] was to build a standard for document exchange between arbitrary systems. ODA specifies three parts for any document:

Logical structure defines the semantic order of a document. Usually, a large text document can be subdivided into chapters and sections.

Layout structure defines the way of presenting the document. There are several possibilities to define spatial positions for the presentation, but the support for temporal synchronization is very restricted.

Content consists of real text modules or other media data. Both logical and layout structure refer to these modules.

ODA mainly defines the static structure of a multimedia document, dynamic aspects such as temporal synchronization are not emphasized in this approach. Special techniques for maintaining consistency between logical structure, layout structure and content are not proposed. The actual data output process is not considered by the ODA standard.

Dexter Reference Model

Initially, the Dexter Model [HS90, HS94] was an architecture for hypertext systems, but later it was used for hypermedia systems, too [SB00]. The Dexter Model consists of the following layers:

The Within-Component-Layer represents the internal data structures which contain atomic data units. The Dexter Model does not make any restrictions on that layer.

The Storage Layer defines the links between several media objects. This layer is very important for the Dexter Model, it specifies several components and connections between them that are necessary for an efficient link tracing.

The Run-Time-Layer manages the presentation of media objects during run-time. There are several possibilities for defining the layout of media objects and spatial relationships between these objects.

It is not possible to define temporal relationships between media objects with the Dexter Model. An enlargement to do so is the Amsterdam Model. It supports temporal relationships that are based on Allen-Relations [All83]. The Dexter/Amsterdam Model can be used to define links as well as spatial and temporal relationships between media objects, but there is no model inherent concept for maintaining consistency of these relationships.

The Dexter/Amsterdam Model can be implemented by using relational database concepts [Spe97]. Several database concepts are very useful for the Dexter Model. As an example links can be realized by foreign keys, thereby, consistency of these links can be guaranteed. However, it is not possible to check consistency for spatial and temporal relationships.

ZyX Model

The ‘ZyX Data Model for Multimedia Documents and Presentations’ [BK99] is a generic model for the presentation of multimedia documents. In contrast to the approaches mentioned above ZyX is built especially for multimedia database systems, a prototype was built on an object-relational multimedia database system [BKW99].

The basis of this approach is a tree that contains several parts of a multimedia document. Within the nodes of that tree spatial coordinates can be defined, thus, spatial relations can be modelled. For temporal relationships special operators are used. However, only a parallel and sequential output is supported. Furthermore, we have *loop* and *delay* as temporal operators.

Even though the ZyX-Model is based on a database system, there is no concept for consistency between defined spatial/temporal synchronizations and stored media objects. In the ZyX-Model media data are stored without considering their output relationships (e.g. synchronization). Therefore, an unfavourable storage structure can lead to problems during a synchronized data output.

Interactive Multimedia Document (IMD)

In contrast to the aforementioned models interactive multimedia documents [MPS⁺00, MPV99, VB97, BKL96] take user interactions with the multimedia document into account. Thus, it is necessary to model spatial and temporal relationships between media data as well as events that arise from user interactions. An interactive multimedia document contains the following points:

Events are the basic elements of an IMD. There are different causes for events, they can arise from user interactions (e.g. mouse clicks) as well as from media objects (e.g. end of a video). Very complex events can be built by means of logical connections of other events. To make a correct reaction on an event the point in time and its position in the presentation is required.

The spatial/temporal composition of multimedia data is essential for modelling an IMD. Usually, constraints are used to define a spatial and temporal order for the presentation.

A scenario describes the behavior of an IMD. We have to define which events can be handled by the IMD. Each event has a list of actions which will be executed if that event occurs. As an example a video has to be stopped if the user presses the stop button.

The transformation (e.g. format transformation, transformation of image size) of media data has to be defined within an IMD. Usually, the kind of transformation and some quality parameters for that transformation are defined, the real transformation algorithm is not part of an IMD.

During a presentation a user can release several events, thus, event sequences arise. Such a sequence shows how a user interacts with the presentation. Mostly, it is not desired that a user can interact with the presentation in an arbitrary way. As an example it can be required that the start of a video is only possible at a certain point during a presentation or that a video can only be stopped after it was started.

From an abstract point of view an event sequence is just a path through the scenario of an IMD. Because we want to control the user behaviour, not all possible event sequences are allowed. Thus, special constraints [MPS⁺00] are used that restrict the possible event sequences. The user behavior depends on the application that uses the interactive multimedia document. This means that different applications have different requirements on the user behavior. Thus, these kinds of conditions can be seen as part of the application logic.

Beside conditions on event sequences we must also define spatial and temporal conditions on media data. This group of constraints builds the real multimedia document from single media data. If we deal with sets of constraints, inconsistencies can occur by constraint modifications. Furthermore, a problem can arise from actions on events. Actions can activate other triggers. This can lead to a not deterministic process if triggers activate each other. The IMD approach only provides consistency checks for temporal constraints between media objects, for this purpose a *temporal constraint network* (TCN) is used. Originally, this technique is used for static constraint checking, but during the output of an IMD the user can make interactions. As an example the user can start or stop a video on arbitrary points in time during a presentation. These user interactions are important for the consistency, they have to be considered by the consistency check which makes this test very complicated.

Actually, the scenario model defines Event-Action-Rules. It is well-known that termination and confluence of a set of Event-Action-Rules cannot be guaranteed. Checking such a set for confluence as well as a test for termination is an NP-hard problem.

Lots of the constraints, defined in an interactive multimedia document, only consider user interaction. These constraints are not important for a multimedia database system itself, because they are part of the application logic and not universal for the stored media data. Of course, the database system must be able to react on certain user actions which occur during the output (presentation) of a multimedia document. To do this user actions must be passed on the database system which must execute an appropriate reaction. As an example a movie is played in a multimedia document, a user wants to stop it and press the stop button in the presentation. The database system must be notified about this event, an adequate reaction of the database system is to stop the video output stream. Usually, user defined functions or methods can be used to implement that reactions within a database system.

Beside application-specific constraints there are some generic constraints for user interactions defined in [MPV99]. An example for these constraints is the general fact that a video can only be stopped after it has been started. These constraints must be guaranteed by database operators. In our example we need operators for starting and stopping videos. The stop operator must check whether the video has been started, so, the operator implements generic constraints for user interactions.

To the best of our knowledge there is no concept for consistency between defined synchronization constraints and stored media data. There is still the problem that a change of media data can modify these data in such a way that the defined synchronization is no more possible. Furthermore, synchronization constraints have no impact on physical data organization. This can lead to problems if a high transfer rate is required during data output [GIÖ98a]. If we want to avoid these problems, an integration of output constraints into the database system is necessary. If the database system maintains output constraints, consistency of the data output can be guaranteed. This means that the stored media data and defined output constraints are always consistent and the database system organizes the physical data structure according to the output constraints.

3.4 Consistency Rules for Multimedia Data

The semantics of multimedia data strongly depends on their presentation. We can distinguish between output processes where user interactions are allowed and those where not. Restrictions at user interactions are part of the application logic. Thus, we do not have to integrate them into a database system. Therefore, we consider only multimedia outputs without user interactions in this work.

There are four groups of consistency rules for multimedia data: rules concerning data quality are the first group, rules dealing either with temporal or spatial relationships and rules for spatiotemporal correlations. In the following we outline each of these groups.

3.4.1 Constraints on Data Quality

As mentioned before, data quality is very important when we have to deal with media data. If we change data quality, it is even possible that the same media data have different meanings in several presentations of a multimedia document. As an example an x-ray image can be displayed with different resolutions. In the case of displaying it with an appropriate resolution we can notice a capillary crack in the bone, with a lower resolution we do not.

In order to avoid this effect, we need constraints that define appropriate output parameters. Usually, the producer of the media data has to define these constraints. However, this kind of constraints can also be used in MOQL [LÖSO97] to build a database query. As an example it is possible to define the output window for images or

videos. Thus, the output resolution is defined implicitly. MOQL is no data definition language. Therefore, it cannot be used for specifying data quality constraints in data definitions.

Constraints for data quality are also known for the realtime data output of media data [MR97, Mar03]. If certain transformation steps are necessary for data transformation, it must be ensured that data quality is maintained in each transformation step. However, the producer of media data cannot define a certain data quality that must be obtained for every data output.

A very formal way for defining constraints on certain data quality parameters is given in [CRS98]. The main idea is using inequalities to define constraints for output parameters. As an example we can define that the difference between the average *throughput* for a whole data stream and the *throughput* for a single media object must be greater as a defined constant. Inequalities can be checked easily, this is the advantage of this approach. On the other side we have the problem that it is hard for the producer of media data to define constraints in such a way. Furthermore, it is not considered how these constraints can be integrated into a multimedia database system.

3.4.2 Temporal Consistency

Usually, media objects have to be presented in a certain temporal order. If the designer of a multimedia document defines such an order, an appropriate data output is expected. Temporal consistency is given when a data output has the defined temporal order.

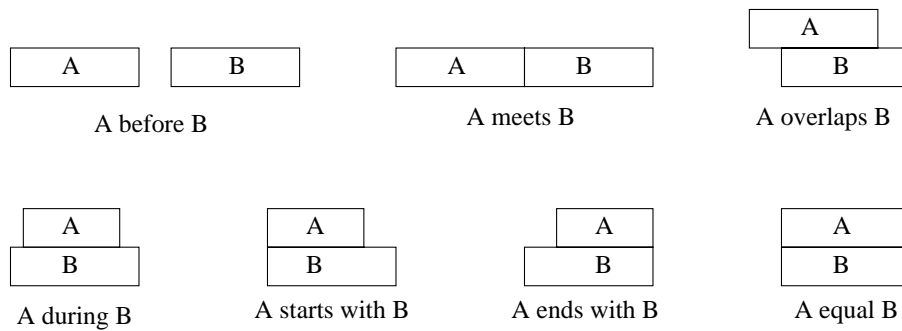


Figure 3.2: Allen-Relations

Allen-Relations [All83] are the most common way of defining temporal constraints between intervals. All possible temporal relations between two temporal intervals can be modelled by using Allen-Relations. An overview of these relations is given in figure 3.2. The temporal intervals are named with *A* and *B*. Except for the *equality* relation, intervals *A* and *B* can be inverted for each relation. Consequently, we have 13 possible Allen-Relations at all.

Inconsistencies can occur when the length of temporal intervals is being changed. Assume we have defined relation *A before B*. Furthermore, we assume that the interval

A has a defined temporal start point. If the length of interval A has been increased, it is possible that interval A meets or overlaps interval B . This means that the originally defined relation *before* is no more possible, because of changing the length of interval A .

Figure 3.3 shows possible inconsistencies in a set of Allen-Relations. Initially, we assume that the data output is consistent to the defined Allen-Relations. Because of modifications of media objects or of Allen-Relations, inconsistencies can occur. The main classes of inconsistencies are *qualitative*, *quantitative* and *indeterminate* inconsistencies [LSI96].

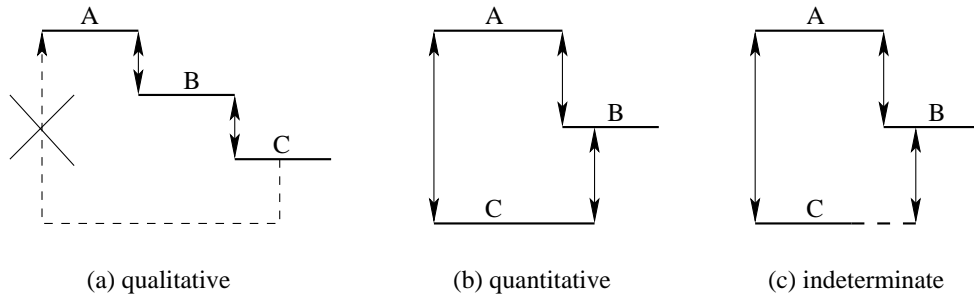


Figure 3.3: Kinds of temporal Inconsistencies [JLR⁺98]

Part (a) of figure 3.3 defines a scenario in such a way that A *meets* B and B *meets* C . Now, it is assumed that the relation C *overlaps* A must be inserted. It is easy to see that such a set of Allen-Relations cannot be consistent. In this case the length of the involved objects has no influence on the consistency of the set. Because of the fact that the set is always inconsistent, it is called a *qualitative* inconsistency.

Part (b) of figure 3.3 defines a temporal order where A *meets* B ; A *starts with* C and C *overlaps* B . This scenario has no qualitative inconsistencies, this means it can be fulfilled under certain conditions. In our example the following conditions must be fulfilled: $length(A) < length(C)$ and $length(A) + length(B) > length(C)$; where $length(X)$ means the length of interval X . Therefore, the relationships in the example are consistent for the following values: $length(A) = 20s$, $length(B) = 15s$, $length(C) = 30s$. Assumed $length(C)$ would be 15 seconds, the defined set of Allen-Relations cannot be fulfilled. In this case the whole scenario would be inconsistent. This set of Allen-Relation can only be fulfilled under certain conditions, therefore, it is *quantitative* inconsistent.

Part (c) of figure 3.3 shows the same scenario as part (b), but it deals with problems caused by user interactions. There are intervals depending on user interactions. This means that a user must do something during the presentation of this interval. As an example a button must be pressed. Because it is unpredictable when the button is pressed, we call this the non-deterministic part of an interval. As an example the length of interval C depends on a user interaction. We use a broken line in interval C to illustrate the non-deterministic part. If we must handle intervals with arbitrary lengths, it is useful to assume a certain maximum and minimum length for that interval. For the

example of part (c) at figure 3.3 we can take again $\text{length}(A) = 20\text{s}$ and $\text{length}(B) = 15\text{s}$. If the scenario should be consistent, $\text{length}(C)$ must be between 20 and 35 seconds.

Without doubt, Allen-Relations are the most common way to model temporal relationships between intervals. The biggest advantage of Allen-Relations is their easy intelligibility. On the other side there are serious problems if a set of Allen-Relations must be checked for consistency. James F. Allen [All83, All84] proposed an algebra and shows how Allen-Relations can be used for reasoning about time. Allen's algorithm is based on a technique called *constraint propagation* which computes all consequences of a new relation using the transitive closure of the temporal relations. Allen proves that his algorithm requires $O(N^2)$ time and space. However, Vilian and Kautz show in [VK86] that Allen's algorithm is not complete. Furthermore, they show that both finding the closer of assertions in the interval algebra and determining the satisfiability of the assertions are NP-hard.

The original Allen-Relations are only qualitative descriptions of relationships between temporal intervals. When we want to define output constraints, a qualitative or functional [DK95] description of interval relations is needed. Functional means that a kind of control structure is defined, as an example the end of an interval defines the start of another interval.

Several other researchers, including Candan et al. [KC96, CPS98, CLS00], Buchanan and Zellweger [BZ93a, BZ93b, BZ93c], Kim and Song [KS95] and Subrahmanian et al. [ASS00, MS96] proposed using a highly-structured class of linear constraints called *difference constraints*. A set of difference constraints can be checked on consistency in a very efficient way. This is the main advantage of that approach. On the other side we have the problem that even simple relationships between intervals lead to complex sets of difference constraints. It is very hard for a user to handle these sets.

If we want to integrate output constraints into a multimedia database system we need both an easy way to define output constraints and an efficient constraint checking method. So, it would be useful if we could use Allen-Relations to define output constraints and use difference constraints to check them. A rough concept for such a transformation is proposed in [CRS98]. This approach only allows defining qualitative relationships between intervals because only the original Allen-Relations are considered.

3.4.3 Spatial Consistency

The spatial order for media objects is also important for multimedia documents or multimedia applications. Usually, the producer of media data or the designer of a multimedia document wants to place data on certain positions. Take the wave field synthesis as example, in this case the sound designer wants to specify that a certain sound source has a specific position in the listening room. If this output constraint cannot be fulfilled, an undesired sound effect is the result and the listener in the cinema room gets a wrong impression.

There are many different approaches to define spatial relationships between spatial objects. Many of these approaches were developed independently from approaches for temporal relationships. So, we have spatial relationships that come from the field of geographical information systems [MGSV99, FGNS00, ES02]. Other approaches come from spatiotemporal logic [GKK⁺03] or directly from constraint database systems [BC98], in this case the mathematical theory of constraint databases is used for shape management.

From the users point of view it is desired to utilize a similar way for defining temporal and spatial output constraints. This is also useful considering constraint checking. If we can use the same formalism for both temporal and spatial output constraints, we are able to use the same techniques for checking the output constraints. Li, Özsu and Szafron [LÖS96] show a possible way to use Allen-Relations also for spatial constraints. The idea behind this approach is to use Allen-Relations separately for each spatial dimension. This means, one Allen-Relation must be defined for each dimension and a conjunction is built from these relations. For the sake of simplicity each salient object is covered by a minimum bounding rectangle. The approach proposed in [LÖS96] is the basis for the definition of spatial output constraints in this work. However, we have to enhance this approach because it only allows qualitative definitions of spatial constraints, but we need quantitative spatial output constraints.

3.4.4 Constraints in Commercial Database Systems

Commercial database management systems, like DB2, as well as free systems, like PostgreSQL, provide no special consistency rules for multimedia data. However, all these systems support integrity rules for static and dynamic integrity constraints. Check-clauses, unique, primary key, and foreign key constraints are supported usually. Trigger and assertions are also known in most systems, however, their usage differs. Some very basic constraints on media data can be built by using traditional integrity rules. For instance an integrity rule can be defined that only allows storing an image if it has a certain resolution. Actually, this kind of constraints is defined on meta-data which are traditional alphanumeric data. Constraints considering the data output cannot be defined in today's commercial database systems.

3.5 Conclusion

This section has shown research results which are the basis of this work. The main conclusions for this work considering existing works are the following:

- Different kinds of applications need data from a multimedia database system. Thus, a list of requirements concerning data types, data access, indexing and format independency can be built. Up to now there are no requirements for data output.

- Integrity constraints are a well-known concept for traditional database systems. Usually, integrity means correct storing of data. Beside the correct data storage media data also need a correct data output, both is very important for their semantics. Maintaining of integrity requires integrity constraints (rules), output constraints are needed for a correct data output (output integrity). Integrity constraints can be defined declaratively, operationally or rule based. For defining output constraints a rule based approach is suitable, because we must define an action for the case of constraint violation. For maintaining integrity we have to verify, translate, optimize, and enforce integrity constraints. All these steps are also necessary for handling output constraints. Existing approaches for multimedia database systems do not support output constraints. Up to now it is common that each database application checks its required media data by its own because there is no possibility for a central check on the database side.
- Several existing approaches of multimedia database systems have been considered. Some works are close to this, but to the best of our knowledge there is no approach for multimedia database systems that deals with constraints on data output. In fact, output constraints can be seen as the missing piece between the storage of multimedia data and existing techniques for format transformation or multimedia database queries.

Chapter 4

Modelling of Output Constraints

The first we need for managing output constraints is modelling output constraints (figure 4.1). For modelling we must consider all classes of output constraints. Thus, a classification of output constraints is required first. However, output constraints can be classified by different criteria (dimensions).

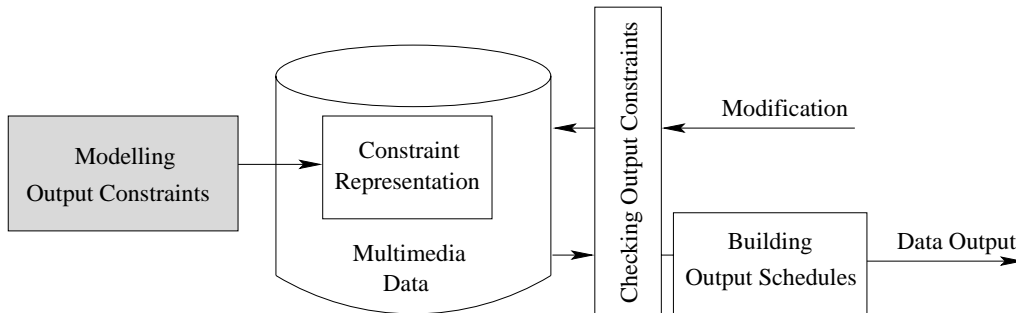


Figure 4.1: Processing of Output Constraints

Thus, this chapter first gives an overview of possible classification criteria of output constraints. Afterwards, a classification is proposed that are used in this thesis. On the basis of this classification we will develop a specification language for output constraints that is suitable for each identified class.

4.1 Taxonomy of Output Constraints

A taxonomy shows what criteria of output constraints can be used for building a classification of output constraints. These criteria are an equivalent of dimensions of a classification. For integrity constraints several classifications are known [SST97, eS97, Tür99, Deß93]. We use these to build a taxonomy for output constraints. Table 4.1 gives an overview of possible classification criteria.

Criterion for Classification	Alternatives	
granularity of data	single output object	
	several output objects	same media type
		several media types
definition	inherent in the model	
	defined explicitly	
output states for checking	static	
	dynamic	
reason for checking	data output	
	data manipulation	insert
		update
		delete
point in time for checking	immediat (after each operation)	
	deferred (at the end of all operations)	
	permanent during the data output	
kind of reaction	disallow the operation	
	transaction abort	
	correcting action	
complexity of checking	complexity classes	
level of tolerance	soft (with range of tolerance)	
	hard (without range of tolerance)	
duration of existence	temporary	
	permanent	
types of output objects	media types	
output feature	output parameter	static
		dynamic
	synchronization	temporal
		spatial
		spatiotemporal

Table 4.1: Taxonomy for output constraints

Granularity of Data: Output constraints can be classified by the size of the data unit which is concerned.

We can build a classification by taking the number of output objects into account that we need for checking the constraint. As an example if we define an output constraint for the frame rate of a video, we have to consider only that video for checking the defined constraint. However, if we define that a video and an audio must output equally, we need both output objects for checking the constraint. If we need more output objects, we can build a group of output constraints that only need output objects of a single media type and a group that deals with output objects of several media types.

Explicit and Inherent Output Constraints: There are several models which can be used for defining output constraints. As an example a lot of synchronization models are well-known [BF98]. Each of these models can express a specific kind of synchronization constraints. Allen-Relations are a synchronization model that supports special key words for temporal synchronizations. Other groups of constraints (e.g. spatial constraints) are not directly supported by this model.

Inherent output constraints are directly supported by the model used for the specification of output constraints. Thus, inherent output constraints are covered by the semantics of the model. *Explicit* or user defined output constraints are not directly supported by the specification model. Each output constraint is either model inherent or explicit. This fact can be used to classify output constraints.

Required Output States for Checking: As above-mentioned the data output for media data is a time depending process. Every single point in time during the data output process has its data output state. So, we have to consider two groups of output constraints. There is a group which only needs one output state for checking. As an example we have defined an output constraint for an image that restricts the minimal resolution for this image. To check this constraint we need only one output state. This is due to the fact that the resolution of the image does not change during data output. Other output constraints need more than one output state for checking. As an example if we define an output constraint for the average delay of frames of a video, we must consider several output states for checking.

We can build a classification on this criterion. An analog classification is known from traditional integrity constraints where static and dynamic integrity constraints exist. It is easy to see that this classification is close to ours of output constraints.

Reason for Checking: The reason for output constraint checking can be an insert, update or delete operation on media data. A database query can also be the reason for checking output constraints. A query leads to the output of media data. Before the data output process can start an output schedule must be built. Furthermore, it is necessary to check output constraints during data output. An appropriate classification can be built that considers the reason for constraint checking.

Point in Time of Checking: Normally, output constraints are checked directly after critical operations (immediate mode). Critical operations are those which modify media data in such a way that output constraints could be violated. There are also complex output constraints which concern many media objects. It is useful to check them after all modification operations (deferred mode). These two classes are very close to traditional integrity constraints. We can also find them in classifications of integrity constraints.

We also have output constraints which must be checked continuously during data output. As an example we have to check the frame rate of a video permanently during output of a video. The output of media data can be seen as database operation. The check of output constraints after finishing this operation is not suitable, rather the output constraints must be checked during this operation.

Kind of Reaction: If output constraints are violated, we can react in several ways. We can classify output constraints by the kind of this reaction. One way to react on output constraint violations is to reject the operation (e.g. an insert operation) or to abort the whole transaction. This may be useful if we check an output constraint after the modification of media data. We can also try to avoid the transaction abort. For this we must correct the media data that violates output constraints. Production rules [CW90] and triggers can be used for that.

If an output constraint is violated during data output, we can abort the whole output process. This can be useful if output parameters like the frame rate of videos gets to worse. An output process needs a lot of resources. So, we waste all these resources if we abort the data output. Instead of an abort it is more suitable to avoid the violation of output constraints. As an example we have defined an output constraint for the minimal frame rate of a video. This constraint must be observed permanently. If it is violated, we can try to tune the system to get a better frame rate.

Complexity of Checking: Checking output constraints requires a certain computational complexity. Each output constraint belongs to a complexity class. These classes can be used to classify output constraints.

Level of Tolerance: We can divide output constraints into *soft* and *hard* output constraints. Hard output constraints do not have a range of tolerance. An example for a hard output constraint is a constraint defining a certain resolution for the output of an image. Soft output constraints have a range of tolerance. This class of output constraints is often used for realizing synchronizations. As an example a video has to start at least 5 time units and at most 10 time units before an audio.

Duration of Existence: Usually, an output constraint is defined for a set of output objects. The constraint is valid for every output object as a whole. As an example if we define an output constraint for a video, it is valid for the entire video. In this case the existence of the output constraint is not limited. The other class of output constraints has a limited existence. As an example we want to define an output constraint for the minimal frame rate of a video. We do not want to define this constraint for the whole video. The output constraint is only needed for the first 5 minutes of the video. Thus, we have an output constraint with a time limit.

It is also imaginable that an output constraint depends on the content of media data. As an example an output constraint could be valid only during a video has specific content. This class of constraints can be activated dynamically.

Types of Output Objects: There are several groups of output objects. We can divide time dependent (e.g. video) and time independent (e.g. image) output objects. Time dependent output objects can be divided into video and audio. It is easy to see that we can find output constraints which can only be defined for specific types of output objects. As an example for a video and an image we can define an output constraint for the minimal resolution. It is not possible to define any resolution for an audio.

Restricted Output Feature: Section 4.2 describes this classification criteria in detail.

4.2 A Classification of Output Constraints

After we have proposed several criteria for classifications in section 4.1 we want to describe the classification that is used in the rest of this thesis (figure 4.2).

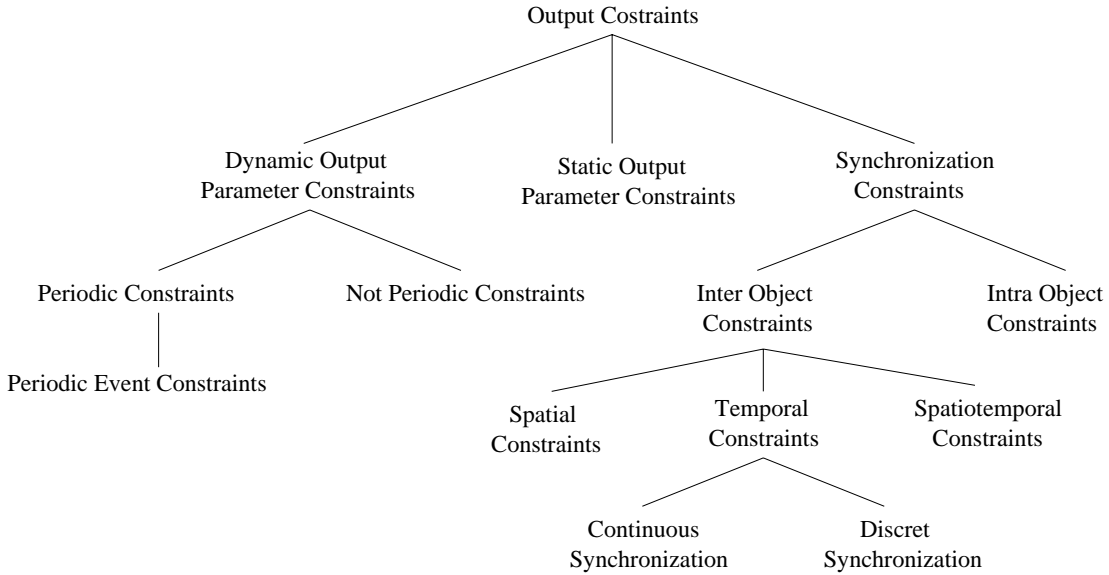


Figure 4.2: Classification of Output Constraints [Hei04b]

As we know from section 4.1 we can use the criteria ‘restricted output feature’ for building a classification of output constraints. Output features are output parameters, like the resolution of an image, as well as several kinds of synchronizations. If we order output constraints according to the output feature they restrict, we will get a classification as shown in figure 4.2.

The main classes of output constraints deal either with output parameters or with synchronization. Considering output parameters we can distinguish between *static* and *dynamic* output parameter. Static output parameters are immutable during the data output, e.g. the resolution of an image. Dynamic output parameters, like the frame rate of a video, are changeable during the output process. The frame rate is also an example of a periodic output parameter because we define it for a specific period, e.g. frames per second. Consequently, we have to introduce classes which deal with periodic output parameters and those which attend non-periodic output parameters.

Output constraints for synchronizations include constraints for intra object synchronization and constraints for synchronizations between output objects. Synchronizations between output objects are realized by spatial, temporal or spatiotemporal output constraints.

We use this classification in this work as the basis for modelling and checking output constraints. For each class a formal specification will be introduced and checking mechanisms will be proposed.

4.3 Fundamentals of Output Constraints

Before we can introduce our specification of output constraints, we must discuss some basics we need for our specification language. First we must mention *temporal logic*. It is needed to define temporal constraints. We will see that temporal logic can be used for spatial constraints as well. After this we will introduce relationships that we need for spatial conditions.

4.3.1 Elements of Constraint Notation

A constraint language is a logical language for expressing restrictions of data and relationships between data. Because of the fact that an output constraint must access data, the constraint language must take the data model into account. In this work it is assumed that the relational data model is used. We use the common notation of this model:

- A relation schema R with degree n is annotated as $R(A_1, A_2, \dots, A_n)$. A_1, \dots, A_n are the attribute names of this relation schema.
- A n -tuple t in a relation $r(R)$ is annotated as $t = \langle v_1, v_2, \dots, v_n \rangle$. The value of the attribute A_i is v_i . With $t.A_i$ the value v_i in t is denoted.
- Usually, the letters R, S are used for relation schemas, r, s denote relation instances and t, u are used for tuple instances.

Usually, first order logic is the basis of a constraint language. First order logic is plain and can be evaluated efficiently. A constraint language consists of logical symbols (e.g.

\exists, \forall), a set of constants, functions, predicates and variables. For our output constraint language we use the same syntax and semantic as the first order logic [Con98]. A constraint language for output constraints must have the following elements:

Symbols are used to connect predicates. We use the same set of symbols as the first order logic: $\{\neg, \vee, \wedge, \rightarrow, \leftrightarrow, \exists, \forall\}$

Variables are necessary to define logical terms like predicates and formulas. Therefore, constraints are defined based on a set of variables. This means for output constraints that the output objects are variables.

Constants are a set of values which can be used in functions or predicates. For our output constraint language, the set of constants is a set of float numbers.

Functions take variables or constants and compute a term. For instance, we assume that we have an audio as an output object o_1 and that we want to determine its loudness. Formally, we can use the function $loudness(o_1)$ to determine the loudness.

Predicates define temporal and spatial relationships between media data. The evaluation of a predicate gives a logical value. The predicates must allow all possible temporal and spatial combinations between media data. On the other side the set should be as small as possible. Furthermore, we need the predicates $\{\leq, \geq, \neq, =\}$. They are necessary if we want to build output constraints for output parameters. As an example we want to restrict the loudness of an output object o_1 . Hence, we can use the predicate \leq and build the term: $loudness(o_1) \leq c$. It is easy to see that $loudness(o_1)$ is a function that takes the variable o_1 . The function determines a specific value for the loudness. The predicate \leq takes this value and the constant c and determines a logical value.

4.3.2 Temporal Logic

Temporal logic is built by adding temporal connectors to a logic language. Explicit references to time are hidden inside the temporal connectors. There are different kinds of temporal logic [Cho94]. Each temporal logic has its own connectors. Several features of temporal logic makes it especially attractive as a query and integrity constraint language for temporal database systems. So, it is possible to define queries and integrity constraints in an abstract, representation-independent manner. The other reason is that temporal logic is amenable to efficient implementation. For example we can translate temporal logic queries to an algebra language.

If we deal with temporal logic there is a choice of *linear* vs. *nonlinear* time. This means time can be seen as a single line or as a tree. Although nonlinear time is potentially applicable to the presentation of multimedia data with user interactions, there has been very little work in this field. User interactions with the database output are

not the scope of this work. Thus, we use *temporal domains* that are defined on linear ordered sets.

Definition 1 (Temporal domain T_P). The structure $T_P = (T, \leq, -, +)$ is a one-dimensional ordered temporal domain. T is a set of temporal instances (e.g. $\{1 \text{ o'clock}, 2 \text{ o'clock}\}$) and \leq is a linear order on T . The function $-$ defines a distance between two elements out of T as follows: $- : T \times T \rightarrow D$; where D is a set of temporal distance elements (e.g. $D = \{1 \text{ hour}, \dots, n \text{ hours}\}$). The function $+$ defines a sum between a point in time and a temporal distance as follows: $+: T \times D \rightarrow T$.

Definition 2 (Distance domain T_D). The structure $T_D = (D, \leq, \cdot)$ is a temporal distance domain. D is a set of temporal distance elements. The relation \leq is a linear order on D . The function \cdot is defined between $i \in \mathbb{Z}$ and $l \in D$ so that $\cdot : \mathbb{Z} \times D \rightarrow D$.

Let $T_P = (T, \leq, -, +)$ be a one-dimensional ordered temporal domain. $D(T)$ is the set $D(T) = \{a - b : a, b \in T\}$ and $D(T) \subseteq D$.

As an example $T_P = (\{0, 1, 2, \dots, 23\}, \leq, -, +)$. The set $\{0, 1, 2, \dots, 23\}$ represents the time instances 0 o'clock, 1 o'clock and so on. A time instance is a concrete point in time. The order \leq is equal to that one which is defined on natural numbers. The function $-$ defines a distance between two points in time. It is easy to see that the difference between two points in time cannot be a point in time again. In reality the difference between two points in time is a span of time. As an example the difference between 9 o'clock and 10 o'clock is 1 hour and never 1 o'clock. Therefore, the function ' $-$ ' maps two points in time into the distance domain T_D . In our example are hours temporal distance elements. The function ' $+$ ' is inverse to the function ' $-$ '. It builds the sum from a point in time and a temporal distance element. The result of this function is a point in time. For instance we build the sum between 9 o'clock and 1 hour. As result we get 10 o'clock, it is a concrete point in time.

As mentioned before the distance domain T_D defines time spans. As an example $T_D = (\{0, 1, 2, \dots, n\}, \leq, \cdot)$ is a distance domain. It is important to note that the elements in the set $\{0, 1, 2, \dots, n\}$ are time spans. The order \leq is equal to that one which is defined on natural numbers. We define a function ' \cdot ', it maps a natural number and a time span into a set of distance elements. As an example we have the distance element 2 hours and the natural number 3. It is easy to see that the result of $3 \cdot 2 \text{ hours}$ is a time span of 6 hours . We can build a set of distance elements $D(T)$ which is suitable to a specific set of time instances T . Thus, for computing $D(T)$ considering our example we have to determine all possible differences between the points in time. All possible time spans between the time points $T = \{0 \text{ o'clock}, 1 \text{ o'clock}, \dots, 23 \text{ o'clock}\}$ are elements of the set $D(T) = \{0 \text{ hour}, 1 \text{ hour}, \dots, 23 \text{ hours}\}$.

Intervals and Periods are very important for the specification of output constraints. We will see that several groups of output constraints are defined on a temporal interval or on a period. As an example if we specify an output constraint which restricts the minimal frame rate of a video during the data output, we have to define the minimal

frame rate as frames per second. This means that we have an output constraint for each second in the video. During the data output we must check that output constraint in each second. The following definitions define intervals and periods formally. Intervals are defined on an interval domain.

Definition 3 (Interval domain T_I [CT98]). Let $T_P = (T, \leq, -, +)$ be a one-dimensional ordered temporal domain. The following set is defined on T :

$$I(T) = \{(a, b) : a \leq b, a \in T \cup \{-\infty\}, b \in T \cup \{\infty\}\}$$

Elements of $I(T)$ are noted as $[a, b]$ and called *intervals*. The following relationships are defined between the elements $[a, b], [a', b'] \in I(T)$:

$$\begin{aligned} ([a, b] <_{--} [a', b']) &\iff a < a' & ([a, b] <_{+-} [a', b']) &\iff b < a' \\ ([a, b] <_{-+} [a', b']) &\iff a < b' & ([a, b] <_{++} [a', b']) &\iff b < b' \end{aligned}$$

The structure $T_I = (I(T), <_{--}, <_{+-}, <_{-+}, <_{++})$ is an interval-based temporal domain T_I .

Definition 4 (Period P). Let $T_P = (T, \leq, -, +)$ be a one-dimensional ordered temporal domain. T_I is an interval-based temporal domain. The following set is defined on $I(T)$ and $D(T)$: $P(I(T), D(T)) = \{([s, e], l) : [s, e] \in I(T), l \in D(T), l \leq e - s\}$. Elements of $P(I(T), D(T))$ are noted as $\langle [s, e], l \rangle$ and called *period*.

Definition 5 (Period elements p_i). If $\langle [s, e], l \rangle$ is a period P then the period elements $p_i \in I(T)$ with $i \in \mathbb{Z}, i \geq 0$ are defined as follows:

$$p_i = \begin{cases} [i \cdot l + s, (i + 1) \cdot l + s] : & (i + 1) \cdot l + s \leq e, \\ & i \cdot l + s \leq e \\ [i \cdot l + s, e] : & e \leq (i + 1) \cdot l + s, \\ & i \cdot l + s \leq e \\ \text{undefined} : & \text{other} \end{cases}$$

$p_i \in P \leftrightarrow p_i$ is defined for P

Informally speaking, a period defined as $\langle [s, e], l \rangle$ starts at point s and ends at point e . A set of period elements p_i is defined for that period. The length of a period element p_i is l . The counter i counts the period elements. As an example we define the period $\langle [1, 4], 2 \rangle$. Period elements are defined for the interval between the start point 1 and the end point 4. Each period element has the length 2. Therefore, the period elements are: $p_0 = [1, 3]$ and $p_1 = [3, 4]$. It is easy to see that the last period element has not the normal time span. This fact must be taken into account if we want to check output constraints which are defined on periods.

4.3.3 Specification of Temporal Constraints

To specify temporal constraints we have the choice of the specification model, which can be based either on time points (instants) or on time intervals. For modelling output

constraints a interval-based specification is natural and thus, we concentrate on it in this work.

As we have seen in section 3 Allen-Relations can define temporal relations between temporal intervals. We can use Allen-Relations for an output constraint language as well, since the several realtionships can be used as predicates that language.

For our application scenarios of section 2 we need a qualitative description of relationships between temporal intervals. For example when we want to model the audio for a movie scene. We exactly have to specify the time between the sound sources. Assuming we want to define that an audio called *violin* must end 2 time units before another audio called *trumpet* starts, we cannot use the original Allen-Relation ‘before’ because it is too unspecific. If we exactly want to define how long one audio source has to start before another, we must restrict the Allen-Relation ‘before’. Consequently, we introduce the relation $before_V(c_1, l)$. We can see from table 4.2 that this relation is used to define a variable time between two temporal intervals. The parameter l of the relation $before_V(c_1, l)$ defines the variability of the time between the two temporal intervals. For our example we can use the relation $violin\ before_V(2, 0)\ trumpet$. In our output constraint language we use restricted Allen-Relations as shown in table 4.2 as temporal predicates.

4.3.4 Specification of Spatial Constraints

Beside temporal constraints we have to specify spatial constraints between output objects. Considering our application scenarios, we have spatial constraints between spatial point objects and between salient objects. A *salient object* [LÖS96] can be an interesting physical object (e.g. car, persons) in a video frame.

Point Objects are adequate for many application scenarios. For example, at wave field synthesis each sound source can be seen as a point in space and we must only handle constraints between these points. Like temporal constraints we want define spatial constraints in a relative manner. This means that we do not want to use the absolute space co-ordinates for spatial constraints. Assume we specify spatial constraints for sound sources in a movie scene. If we used the absolute coordinates of the cinema room, we would have to specify the spatial constraints for several cinema rooms. Therefore, it is more appropriate for our applications to specify spatial constraints relatively to the center point of the cinema room.

To simplify matters we concentrate on spatial constraints in a 2-dimensional space. In spatial constraints we must define qualitative and quantitative relationships between points in space. Therefore, we have to specify the direction and the distance between the points in space. Table 4.3 shows all spatial constraints we use for point objects. If we want to support spatial constraints between point objects in a higher-dimensional space, we have to introduce spatial constraints for each further dimension. We can reach the same expression power with a smaller set of constraints. We only need one

$A \text{ before}(c_1) B$	The end of A is at least c_1 time units before the start of B ; with $c_1 > 0$
$A \text{ before}_V(c_1, l) B$	Variable time interval $[c_1, (c_1 + l)]$ between the end of A and the start of B ; with $c_1 > 0, l \geq 0$.
$A \text{ overlaps}(c_1, c_2) B$	The time between the start of A and the start of B is at least c_1 . The time between the end of A and the end of B is at least c_2 ; with $c_1 > 0, c_2 > 0$.
$A \text{ overlaps}_{V_s}(c_1, l, c_2) B$	Variable time interval $[c_1, (c_1 + l)]$ between the start of A and the start of B . The time between the end of A and the end of B is at least c_2 ; with $c_1 > 0, c_2 > 0, l \geq 0$.
$A \text{ overlaps}_{V_e}(c_1, c_2, l) B$	Variable time interval $[c_2, (c_2 + l)]$ between the end of A and the end of B . The time between the start of A and the start of B is at least c_1 ; with $c_1 > 0, c_2 > 0, l \geq 0$.
$A \text{ overlaps}_{V_{se}}(c_1, l, c_2, m) B$	Variable time interval $[c_1, (c_1 + l)]$ between the start of A and the start of B as well as between the end of A and the end of B $[c_2, (c_2 + m)]$; with $c_1 > 0, c_2 > 0, l \geq 0, m \geq 0$.
$A \text{ during}(c_1, c_2) B$	The time between the start of A and the start of B is at least c_1 . The time between the end of A and the end of B is at least c_2 ; with $c_1 > 0, c_2 > 0$.
$A \text{ during}_{V_s}(c_1, l, c_2) B$	Variable time interval $[c_1, (c_1 + l)]$ between the start of A and the start of B . The time between the end of A and the end of B is at least c_2 ; with $c_1 > 0, c_2 > 0, l \geq 0$.
$A \text{ during}_{V_e}(c_1, c_2, l) B$	Variable time interval $[c_1, (c_1 + l)]$ between the end of A and the end of B . The time between the start of A and the start of B is at least c_1 ; with $c_1 > 0, c_2 > 0, l \geq 0$.
$A \text{ during}_{V_{se}}(c_1, l, c_2, m) B$	Variable time interval $[c_1, (c_1 + l)]$ between the start of A and the start of B . Variable time interval $[c_2, (c_2 + m)]$ between the end of A and the end of B ; with $c_1 > 0, c_2 > 0, l \geq 0, m \geq 0$.

Table 4.2: Restrictions for Allen-Relations [Hei05]

constraint for each dimension. It is easy to see that $\text{pointA left}(c, l) \text{ pointB}$ is equal to $\text{pointB right}(c, l) \text{ pointA}$. The constraint *spatial-equal* can also be replaced by a combination of others. We support all constraints of table 4.3 which makes the modelling of spatial constraints easier for the user.

<i>pointA west(c.l) pointB</i>	Variable spatial interval $[c, (c+l)]$ between <i>pointA</i> and <i>pointB</i> . <i>pointA</i> is west of <i>pointB</i> ; with $c > 0, l > 0$.
<i>pointA east(c.l) pointB</i>	Variable spatial interval $[c, (c+l)]$ between <i>pointA</i> and <i>pointB</i> . <i>pointA</i> is east of <i>pointB</i> ; with $c > 0, l > 0$.
<i>pointA south(c.l) pointB</i>	Variable spatial interval $[c, (c+l)]$ between <i>pointA</i> and <i>pointB</i> . <i>pointA</i> is south of <i>pointB</i> ; with $c > 0, l > 0$.
<i>pointA north(c.l) pointB</i>	Variable spatial interval $[c, (c+l)]$ between <i>pointA</i> and <i>pointB</i> . <i>pointA</i> is north of <i>pointB</i> ; with $c > 0, l > 0$.
<i>pointA spatial-equal pointB</i>	<i>pointA</i> and <i>pointB</i> have the same spatial position.

Table 4.3: Spatial Predicates for Point Objects

Salient Objects like persons or cars in a movie scene must be represented in a way that allows defining relationships between these objects. A common way is building an approximation of salient objects. Usually, a minimum bounding rectangle (MBR) with boundaries parallel to the horizontal and vertical axis of a coordinate system is built.

Definition 6 (Bounding Box [LÖS96]). The bounding box of a salient object A_i is defined by its MBR (A_{ix}, A_{iy}) and a depth A_{iz} where $A_{ix} = [x_{s_i}, x_{f_i}]$, $A_{iy} = [y_{s_i}, y_{f_i}]$, $A_{iz} = [z_{s_i}, z_{f_i}]$. x_{s_i} and x_{f_i} are A_i 's projection on the X axis with $x_{s_i} \leq x_{f_i}$ and similarly for $y_{s_i}, y_{f_i}, z_{s_i}$ and z_{f_i} .

Spatial relations between salient objects are modelled as spatial relations between the MBR's of these objects. Between two regions eight meaningful relations have been identified which lead to eight predicates called *equal*, *disjoint*, *coveredBy*, *covers*, *overlap*, *meet*, *inside*, *contains*. For defining output constraints a similar model of temporal and spatial relations is needed. Therefore, we want to use Allen-Relations for defining spatial relationships as well. Our approach is based on [LÖS96]. J.Z. Li, T. Özsu and D. Szafron have introduced sets of strict directed relations, mixed directed relations and positional relations between bounding boxes. They have also shown, how these spatial relations can be defined in terms of Allen's relational algebra. We cannot directly use these spatial relationships in this work, because we need a qualitative specification for spatial relationships. The basic idea proposed in [LÖS96] is that each spatial dimension is considered separately. Hence, in a two-dimensional space we must consider the X and Y axis separately. Therefore, the A_{ix} and the A_{iy} sites of MBR's can be seen as intervals in an one-dimensional space and Allen-Relations can be used for defining relationships. We build one predicate based on Allen-Relations for each spatial dimension and the conjunction of these predicates is the spatial condition.

Figure 4.3 shows two MBR's which are in a spatial relation called *touch* [LÖS96]. We have made a projection of A_{ix} and A_{iy} on the axis, now the accompanying Allen-Relations for each dimension can easily be seen. In X direction we have A_{1x} *meets* A_{2x} and in Y direction we have A_{1y} *starts* A_{2y} . If we put both dimensions together, we

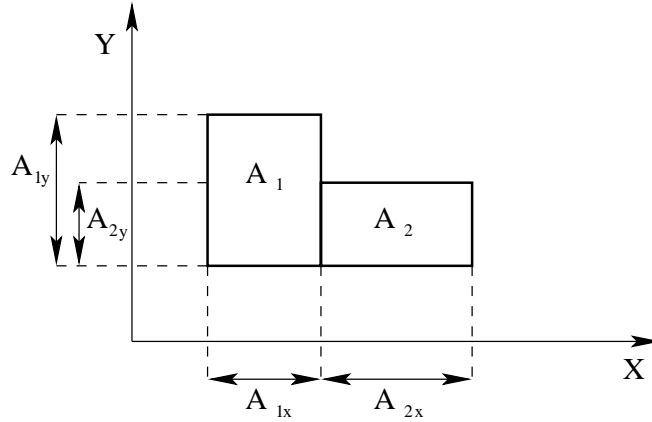


Figure 4.3: Spatial Condition Using MBR's of Salient Objects

get $(A_{1x} \text{ meets } A_{2x}) \wedge (A_{1y} \text{ starts } A_{2y})$. This basic example does not show restricted Allen-Relations, but it is easy to see that they can be used in the same manner.

4.4 The Database Output

Our output constraints are rules to define a correct data output of media data. Therefore, we must consider the database output in more detail. We assume that each stored media object can be output. Thus, a set of *output objects* $OUTOBJ$ exists which can be built from the stored media objects. The stored media data and the output objects belong to them can be very different. The reason for this is the format independency which is required by a multimedia database system.

As mentioned before we use the relational data model in this work. Thus, we assume that media objects, like videos, are stored as attribute values of a relation. Usually, beside the real media object (e.g. video) metadata (e.g. producer of the video) of media objects are stored in the database as well. Without loss of generality, we assume that a media object (mo) and its metadata are stored in a single tuple. Thus, we use in the following the relation schema $R(A_1, A_2, \dots, A_n, A_{n+1}^{MO}, \dots, A_m^{MO})$. A_x^{MO} are names of attributes that store media objects. The domain of these attributes is a set of media objects called MO .

An output function f_{out_j} builds an output object o from a stored media object $mo \in MO$.

$$f_{out_j} : MO \rightarrow OUTOBJ_j, f_{out_j}(mo) = o \quad (4.1)$$

Different output functions are possible on the same stored media object. Every output function has its own domain. As an example a text is stored as a media object. One output function can build an audio as output object from that text. Another output function can build an image (e.g. PDF) as output object from the same text. The domains of both output functions are very different. Different output constraints can be used to

restrict these domains of output objects. Considering our example the resolution can be restricted for the image but not for the audio. We consider the following groups of output functions:

- $f_s(mo)$ produces spatial output objects
- $f_{td}(mo)$ produces time dependent output objects
- $f_{s/td}(mo)$ produces spatial and time dependent output objects

A *spatial output object* has no internal time dependency. An image is an example for this group of output objects. It has only a spatial shape but it is not time dependent. *Time dependent output objects* (e.g. audio) do not have any spatial shape but they have an internal time dependency. A video is an example for *spatial and time dependent output objects*. It is easy to see that it has a spatial shape and an internal time dependency.

Equation 4.2 shows a formal definition for the domain of an output function. All output objects o produced by the function $f_{out_j}(mo)$ are members of this domain. The co-domain MO of the output function is a set of media objects.

$$OUTOBJ_{f_{out_j}}^{MO} = \{o \mid o = f_{out_j}(mo) \wedge mo \in MO\} \quad (4.2)$$

The domain of all possible output objects for a set of media objects MO is called $OUTOBJ^{MO} = \{f_{out_j}(mo) \mid 1 \leq j \leq x \wedge mo \in MO\}$. Output constraints restrict the sets $OUTOBJ_{f_{out}}^{MO}$ in order to define legal output objects. The general form of an output constraint is shown in equation 4.3.

$$\{\forall, \exists\} o \in OUTOBJ_{f_{out}}^{MO} : F \quad (4.3)$$

F is a first order logic formula but in this formula we can also use the temporal and spatial predicates, which we have introduced. The meaning of this definition is that for one or for all output objects, built from the set of media objects MO , the formula F must be true.

Most output constraints restrict all elements in a set of output objects. Therefore, output constraints often have the form: $\forall o \in OUTOBJ_{f_{out}}^{MO} : F$. Thus, we use in the following the notation of equation 4.4 to denote that the predicate F must be fulfilled for each element in $OUTOBJ_{f_{out}}^{MO}$.

$$OUTOBJ_{f_{out}}^{MO} \div F \quad (4.4)$$

4.5 General Structure of Output Constraints

For a general specification of integrity rules (section 3.2.2) we have to define the constraint name, the time point of constraint checking, the condition clause, and a reaction. Thus, we have to consider all these parts in our general structure of output constraints:

The Output Constraint Name is required if we want to handle this constraint as a database object.

The Reason for Checking defines when the output condition is checked. This check can be executed after modifications on stored data or before data output.

The Output Condition defines a term that must be fulfilled for the data output process. For defining output conditions we need special predicates to specify temporal and spatial conditions. Thus, the predicates of tables 4.2 and 4.3 must be supported.

In practice, some SQL extensions support temporal or spatial predicates. For example TSQL supports the following temporal predicates [Sno00]:

$A \text{ PRECEDES } B$ implements $A \text{ before } B$
 $A \text{ SUCCEEDS } B$ implements $A \text{ before}^{-1} B$
 $A \text{ MEETS } B$ implements $A \text{ meets } B \vee A \text{ meets}^{-1} B$
 $A \text{ CONTAINS } B$ implements $A \text{ during } B \wedge A \neq B$
 $A \text{ OVERLAPS } B$ implements $A \text{ overlaps } B \vee A \text{ overlaps}^{-1} B \vee A \text{ starts } B$
 $\vee A \text{ starts}^{-1} B \vee A \text{ finishes } B \vee A \text{ finishes}^{-1} B$
 $\vee A \text{ during } B \vee A \text{ during}^{-1} B \vee A \text{ equals } B$

However, these predicates can only test the qualitative relationships between *periods*. TSQL supports *periods*, which define a temporal interval with a fixed start and end point. Actually, this periods come from temporal database systems. Hard coded start and end points are required for defining these periods. Therefore, we cannot use them for output constraints.

The Reactions on condition violations must be defined for two cases. We need a reaction that must be executed if the output condition is violated by modifications on stored media data. Another reaction defines what should be done if the output condition is violated during data output. Both reactions are normally defined by the user. This is necessary because a standard reaction is not convenient. So, we cannot say that an abort of a data output is always suitable when an output condition is violated.

The name of an output constraint as well as the reason for checking can be specified directly (section 4.7). The reaction of a condition violation is either a basic reaction, like a transaction abort, or a complex operations, like a program call. Thus, an elementary specification can be used for defining the reaction (section 4.7). However, specifying output conditions needs more attention. We need an uniform definition of output conditions that supports all classes of output constraints. Thus, we will consider the specification of output conditions in detail in the following sections.

4.6 Specification Language for Output Conditions

This section shows how formal specifications for several classes of output conditions can be built. We use the classification of figure 4.2 for the following considerations.

We will introduce an example that is used to explain the formal specifications. Our example is based on the application which deals with wave field synthesis for cinemas. Each movie scene has a number of sound sources which are audible in the scene. We assume that the following table exist in the database:

$$MOVIE(SCENE_NR, SOUND_SOURCE_1, \dots, \\ SOUND_SOURCE_N, MOVIE_SCENE)$$

Such a table exists for each movie handled by the database. This table stores all scenes for a certain movie and all sound sources belong to the scenes. For the sake of simplicity we assume that all sound source attributes contain real audio data. The attribute *MOVIE_SCENE* contains the videos to the scenes.

4.6.1 Conditions for Static Output Parameters

Static output parameters are those which do not change during the output process of output objects. For our movie example the resolution of the film is a static output parameter. In practice cinemas have different screen sizes. Therefore, different resolutions of a film must be provided. The resolution is a static output parameter because it usually does not change during the movie presentation.

Equation 4.5 shows the formal definition of constraints for static output parameters. The function $p(o)$ determines the value of an output parameter in an output object (e.g. resolution of a video). c is a constant and $\theta \in \{\leq, \geq, =\}$.

$$OUTOBJ_{f_{out}}^{MO} \div p(o) \theta c \quad (4.5)$$

Equation 4.5 can be transformed into the general form of output condition. The term $p(o) \theta c$ can be written as first order logic predicate $\theta(p(o), c)$. Thus, we can transform equation 4.5 into the general form of output conditions (equation 4.3) as follows: $OUTOBJ_{f_{out}}^{MO} \div \theta(p(o), c)$.

Equation 4.6 shows an example of an output condition on a static output parameter. We restrict all output objects o which are built by a spatial output function f_s from videos stored in the attribute *MOVIE_SCENE* in relation *MOVIE*. The horizontal resolution (denoted by *hor_resolution*) of each output object must be greater than or equal to 1365. Only output objects which satisfy this constraint should be output.

$$OUTOBJ_{f_s}^{\{z.MOVIE_SCENE \mid movie(z)\}} \div hor_resolution(o) \geq 1365 \quad (4.6)$$

The following expression shows the same output condition. Instead of the compact notation of the output domain *OUTOBJ* we use, according to equation 4.2, a more detailed notation that is easier to understand.

$$\{o \mid o = f_s(mo) \wedge mo \in \{z.MOVIE_SCENE \mid movie(z)\}\} \\ \div hor_resolution(o) \geq 1365$$

Other examples for output constraints on static output parameters can be found in our hospital scenario. In the hospital x-ray images of patients must be handled. Static output parameters like resolution are very important for these media data. Therefore, the producer of an x-ray image must define a certain output resolution.

Because output constraints only restrict output objects, we can store media data, which do not fulfill the defined output constraints, into the database system. This is useful if we only have a movie scene or an x-ray image with a low resolution and no chance to get one with the required resolution. In this case we can store the data we have.

4.6.2 Conditions for Dynamic Output Parameters

As we know from section 4.2 dynamic output parameters are distinguished into periodic and non-periodic output parameters. Thus, we want to consider output conditions for both types of parameters in the following.

Non-Periodic Dynamic Output Parameters are defined for a temporal interval during the data output. Assume we define an output condition for the loudness of a sound source. During the presentation of a movie in a cinema it is not desired to regulate the loudness of sound by hand. The author of the movie sound must define the loudness during the modelling process. We have to store this control information together with the audio data into the database system. The system must control the loudness during the data output. It is easy to see that the loudness of sound sources is a dynamic output parameter because it often changes during the presentation of a movie scene. If we want to define a specific loudness for a certain interval of the sound source during the data output, we can define an appropriate output condition.

Each time dependent media object has an internal media time. The internal media time is defined by an attribute that has a temporal domain. For example, the frame numbers of a video define such an internal media time. A time dependent output object also has an internal time. We assume that a mapping exists between the internal time of the stored media object and the time of the output object. For simplicity, it is assumed that an output object has the same internal media time as its multimedia object.

A condition can then be defined for a temporal interval $[s, e]$ of the internal media time. To describe the end of the internal media time the symbol ∞ is used. For all points t of the interval the defined condition must be met. Equation 4.7 gives a formal definition for conditions of dynamic output parameters that are not periodic. The defined condition must be fulfilled for each output object o of an output object set. The function $p(t, o)$ determines the value of an output parameter of an output object at a point in time t (e.g. loudness of a sound source with sample number t). c is a constant and $\theta \in \{\leq, \geq, =\}$.

$$OUTOBJ_{ftd}^{MO} \div (\forall t \in [s, e] : p(t, o) \theta c) \quad (4.7)$$

To give a practical example we want to use the movie example again. We want to define an output condition only for the output objects of a certain sound source. Let us assume the output condition is supposed to restrict *sound_source_1* in scene ‘2’. For all output objects generated from this audio and for all points in time from 1 to 100 the defined condition must be fulfilled. The output constraint defines that the loudness must be greater or equal 20 decibel. This means, that a user cannot define another value for the loudness for the time interval from 1 to 100. The expression below shows the output condition for our example.

$$\begin{aligned} & \{o \mid o = f_{td}(mo) \wedge mo \in \\ & \quad \{z.SOUND_SOURCE_1 \mid movie(z) \wedge z.SCENE = '2'\} \\ & \} \div (\forall t \in [1, 100] : loudness(t, o) \geq 20) \end{aligned}$$

Only output objects which satisfy this constraint should be output. The constraint checking is only possible during the output process. Of course, the stored audio signal has a certain loudness. We could restrict this loudness by an integrity constraint (assertion) which we can generate from our output condition. However, we can change the loudness of a stored audio signal arbitrarily by an output function. Therefore, we have to check the condition for the output object and not for the stored data.

It is very costly to check the output condition for each point in time. We need some more feasible strategies for checking output conditions. One strategy is to check random samples during the interval. It is also possible to check the output conditions in certain time steps.

Periodic Output Parameters are those which are defined on a period. The frame rate of a video is an example for this group of output parameters. We can use periodic output conditions to restrict these output parameters. Periodic output conditions are defined on a period as the output parameters, too. We assume that the period defined in the output condition and the period of the parameter have the same length. For instance, if the frame rate of a video is 25 frames per second the period of the parameter is one second. Therefore, a condition that observes this parameter must be checked every second.

Equation 4.8 shows the formal definition of periodic conditions for dynamic output parameters. The condition is defined for the output objects $o \in OUTOBJ_{f_{td}}^{MO}$. The output function f_{td} produces timedependent output objects. Thus, this group of output conditions can only restrict output objects with an internal time dependency. In a periodic condition we define the period $\langle [s, e], l \rangle$ on the internal time of the output object. The interval $[s, e]$ defines the start and end point of the period and l is the length of a time interval, it defines the length for each period element. Take as an example the period definition $\langle [1, \infty], 2 \rangle$. There the period starts at the first time unit and ends with the last time unit of the output object. The length of each period element p_i is two time units.

The condition should be checked for each period element p_i of a period $\langle [s, e], l \rangle$. The function $u(p_i, o)$ determines the value of the restricted output parameter of an output object for each period element p_i . This means if we want to restrict the frame rate, a function is needed that determines the frame rate for each second. c is a constant and $\theta \in \{\leq, \geq, =\}$.

$$OUTOBJ_{f_{td}}^{MO} \div (\forall p_i \in \langle [s, e], l \rangle : u(p_i, o) \theta c) \quad (4.8)$$

For example we want to define an output condition for a periodic output parameter on our *MOVIE* relation. The frame rate of a video is a periodic output parameter and we want to define an output condition for all output objects produced by the video of scene ‘2’. For these output objects the frame rate should be greater or equal to 25 frames per second. This output condition must be valid from the first time unit of the output object and must be fulfilled until the end of the output. The duration of a period element defined in the output condition must be equal to the period of the dynamic output parameter. Thus, we have defined one time unit (second) as duration of period elements. The following shows the output condition according to our example:

$$\{o \mid o = f_{td}(mo) \wedge mo \in \{z.MOVIE_SCENE \mid movie(z) \wedge z.SCENE_NR = '2'\} \\ \} \div (\forall p_i \in \langle [1, \infty], 1 \rangle : frame_rate(p_i, o) \geq 25)$$

In other words we define for each time dependent output object, produced from a certain video, that for each defined period element the frame rate is greater or equal to 25 frames per second. To determine the frame rate we use the function $frame_rate(p_i, o)$ that takes a period element and the output object as input. With the predicate \geq we achieve a logical value. It is easy to see that this output condition can only be checked during the data output process. The condition must be checked for each period element, i.e. we have a periodic condition check.

Periodic Event Conditions are a more specific group of periodic output conditions. They restrict sequences of events within periods. In some cases it is very important to determine events which occur during data output in a certain period element. An example for this are *jitter-constrained periodic event streams*. In these streams certain events occur in a semi-fixed span. This is the reason why very efficient buffer strategies can be used for these streams. However, we can only use these strategies if we know that the output object is a jitter-constrained periodic event stream. Until now there is no suitable way to detect whether a stream is a jitter-constrained periodic stream or not.

However, our approach of *periodic event conditions* can be used to check whether or not a stream has the properties of a *jitter-constrained periodic event stream* [Ham97]. Such a stream has the following parameters:

- $L > 0$ average event distance, i.e. the length of the period
- $0 \leq D \leq L$ minimum distance between successive events
- $\tau > 0$ maximum lateness for an event (relative to the end of its period).

The parameters D, τ and L are fixed. The event E_i occurs at point in time a_i . For a jitter-constrained periodic stream, it must be ensured that the distance between events is nearly constant. Formally this is written as $\forall E_i : a_i < i \cdot L + \tau \wedge a_{i+1} - a_i \geq D$.

Equation 4.9 shows the formal definition of periodic event conditions. It is defined for the time dependent output objects $o \in OUTOBJ_{f_{td}}^{MO}$. The condition should be checked for each period element p_i of a period $\langle [s, e], l \rangle$. $\langle E \rangle_i^o$ is a sequence of events. This notation means that the sequence of events belongs to the period element p_i of the output object o . In other words the events are generated from the output object o during the period element p_i . c is a constant and $\theta \in \{\leq, \geq, =\}$.

$$OUTOBJ_{f_{td}}^{MO} \div (\forall p_i \in \langle [s, e], l \rangle : u(\langle E \rangle_i^o) \theta c) \quad (4.9)$$

Assume we want to check whether a stream is a jitter-constrained stream or not. We can use the *MOVIE* relation again for an example. The output condition should work on every time dependent output object generated from a stored video. As long as the output condition is fulfilled the output object is a jitter-constrained stream. So, we can use special optimization strategies for the database buffer. If the output condition is violated, an adequate action must be executed, this means the buffer strategy must be changed. In the following example we assume that the span between two events (L average event distance) takes 5 time units and the condition should be valid for the whole stream ($\langle [1, \infty], 5 \rangle$). As we know from equation 4.9, we need a predicate that checks the properties of a jitter-constrained stream. In the example below this predicate is called *jitter_constraint*. If we want to check those properties, we must take the time points of the event occurrence into account. Therefore, the predicate *jitter_constraint* takes a sequence of time points as parameters. This is possible in that case because events and time points are synonyms. The predicate evaluates the following expression: $a_{j+1} - a_j \geq D \wedge a_j \leq j \cdot L + \tau$. The terms D, τ and L are fix and known.

$$\{o \mid o = f_{td}(mo) \wedge mo \in z.MOVIE_SCENE \mid movie(z)\} \div (\forall p_i \in \langle [1, \infty], 5 \rangle : (jitter_constraint(\langle event \rangle_i^o))$$

Now we want to use a periodic event condition to express the same condition as in the example for periodic output conditions. This means, we want to restrict the frame rate for all time dependent output objects generated from the stored video of scene '2'. The frame rate for these output objects should be greater or equal to 25 frames per second. Every output of a frame can be seen as an event. If we define the output of a frame as event, we can say that we require 25 such events for each second. Thus, we can easily count this events for each second. In the output condition below each period element p_i has a duration of one second. The sequence $\langle frame_output \rangle_i^o$ represents the frame output events which have arisen during the period p_i . The function *count* counts these events and the predicate \geq gives a logical value.

$$\{o \mid o = f_{td}(mo) \wedge mo \in \{z.MOVIE_SCENE \mid movie(z) \wedge z.SCENE_NR = '2'\} \} \div (\forall p_i \in \langle [1, \infty], 1 \rangle : count(\langle frame_output \rangle_i^o) \geq 25)$$

4.6.3 Synchronization Conditions

In our classification of output constraints (section 4.2) we have identified intra-object and inter-object synchronization constraints. Considering output constraints inter-object synchronization and its subclasses temporal, spatial, and spatiotemporal output constraints are very important. Thus, this section concentrates on them.

Intra-Object Conditions

Intra-object conditions define synchronization relationships within an output object. As an example the temporal order of frames in a video is an intra object conditions. Intra-object conditions define the structure of the output objects. Usually, this structure is defined by the data format (e.g. MPEG). Therefore, the producer or the user of media data cannot influence intra-object synchronization. From reasons of the completeness intra-object conditions are also mentioned in this work.

Inter-Object Conditions

This group of output condition can be used to define the output of complex multimedia documents. Up to now it is not possible to define temporal and spatial inter-object conditions into a multimedia database system. Thus, the multimedia database system cannot maintain the consistency of complex multimedia documents. This means, the database system is neither able to ensure the semantic integrity nor can it ensure the output integrity.

With inter-object conditions the producer of multimedia data can specify the order of the data output. In our movie example we can define temporal output constraints between the sound sources. When the duration of sound sources is changed, modifications on sound sources can lead to violations of these constraints. The database must maintain the defined synchronization conditions in the case of data modifications and during data output.

Spatial Output Conditions define a spatial order between output objects. As an example we want to take a spatial condition which is needed in the wave field synthesis in our movie example. Each scene has a set of sound sources which must have a spatial order during the presentation of the movie scene. To simplify matters we assume that each sound source does not change its spatial position during the scene. Therefore, the sound master can describe the scenes *statically*, this means the time is not important for the output conditions. Figure 4.4 shows two sound sources of a movie scene. For wave field synthesis each sound source is a point object in the listening room, with a spatial output condition we can define the distance between sound sources as it is shown in figure 4.4. It is assumed that both sound sources exist during the whole scene. This means, even when a sound source does not play it exists, but it is in silence.

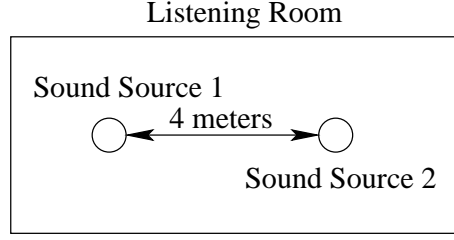


Figure 4.4: Spatial Output Condition for Sound Sources

Equation 4.10 shows the formal definition of spatial output conditions. These conditions only work for spatial output objects. Thus, we need an output function which generates either a spatial or spatiotemporal output object. With condition 4.10 we define that for each spatial output object o_1 there must exist a spatial output object o_2 which fulfills the spatial predicates. Assuming that we define the spatial condition for point objects in a 2-dimensional space, we can use the spatial predicates *west*, *east*, *south*, *north*, *spatial-equal* (table 4.3). If we want to define spatial conditions between salient objects, we must use the spatial predicates (section 4.3.4) for this group of objects.

$$\begin{aligned} \forall o_1 \in OUTOBJ_s^{MO_1} : \exists o_2 \in OUTOBJ_s^{MO_2} : \\ spatial_predicate(o_1, o_2) \end{aligned} \quad (4.10)$$

The following example defines the spatial output constraint shown in figure 4.4 formally. For each spatial output object o_1 built from an audio stored as attribute value in *SOUND_SOURCE_1* of scene ‘1’ a spatial output object o_2 must exist which comes from the same scene, but it must be built from the audio in *SOUND_SOURCE_2*. Both output objects (sound sources) are point objects, thus we can define with o_1 *west*(4.0) o_2 that o_1 is exact 4 space units west from o_2 .

$$\begin{aligned} \forall o_1 \in \{o \mid o = f_s(mo) \wedge \\ mo \in \{u.SOUND_SOURCE_1 \mid movie(u) \wedge SCENE_NR = '1'\}\} : \\ \exists o_2 \in \{o \mid o = f_s(mo) \wedge \\ mo \in \{u.SOUND_SOURCE_2 \mid movie(u) \wedge SCENE_NR = '1'\}\} : \\ (o_1 \text{ west}(4.0) o_2) \end{aligned}$$

Temporal Conditions define a temporal order between time dependent output objects. Thus, the applied output function must produce output objects with an internal time. Therefore, we can only use output functions which produce time dependent or spatial and time dependent output objects.

Equation 4.11 shows the formal definition of a temporal output condition. The condition defines that for each time dependent output object o_1 must exist a time dependent output object o_2 which fulfills the temporal predicate. For our condition language we must use restricted Allen-Relations of table 4.2 as temporal predicates.

$$\begin{aligned} \forall o_1 \in OUTOBJ_{td}^{MO_1} : \exists o_2 \in OUTOBJ_{td}^{MO_2} : \\ temporal_predicate(o_1, o_2). \end{aligned} \quad (4.11)$$

Figure 4.5 shows that two sound sources must be output equally. This temporal output condition must be fulfilled for sound sources of a certain movie scene. For our movie

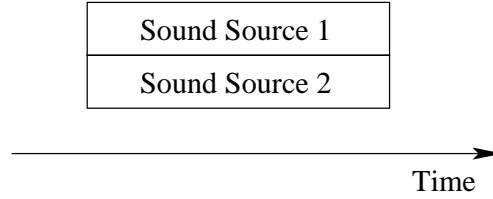


Figure 4.5: Temporal Output Condition for Sound Sources

example we can use the following temporal condition to define the output condition shown in figure 4.5:

$$\begin{aligned} \forall o_1 \in \{o \mid o = f_{td}(mo) \wedge \\ mo \in \{u.SOUND_SOURCE_1 \mid movie(u) \wedge SCENE_NR='1'\}\} : \\ \exists o_2 \in \{o \mid o = f_{td}(mo) \wedge \\ mo \in \{u.SOUND_SOURCE_2 \mid movie(u) \wedge SCENE_NR='1'\}\} : \\ (o_1 \text{ equal } o_2) \end{aligned}$$

By using the Allen-Relation *equal* we can express that both output objects have the same start and end time points. The synchronization is only defined on the output objects as a whole and the condition check only takes these time points into account. It is possible that the temporal condition is fulfilled for the start and end time points of the output object, but during the output of the output objects the synchronization condition is not satisfied. Assume we want to define a lip synchronization for the output of a video and an audio. Now it is no more sufficient to consider only the start and end time points of the output objects. We must define and check the synchronization condition on smaller units.

It is often useful to define a fine granular synchronization between time dependent output objects. To achieve this, we must define synchronization conditions for small parts of the time dependent output objects. Small parts of a time dependent output object can be generated by defining a period with short period elements on the output object. Hence, we define the period $\langle [s, e], l \rangle^{o_i}$ on the output object o_i . The start time point of this period is s and its end is at time point e . A period element has the length l . Equation 4.12 uses periods of time dependent output objects and defines temporal conditions on these periods. For every period element p_i of output object o_1 exists a period element q_i of output object o_2 . The temporal predicate takes the period elements p_i and q_i as input. The temporal condition is fulfilled when the temporal predicate is

true for each period element. As temporal predicate we can use each of the restricted Allen-Relations shown in table 4.2.

$$\begin{aligned} \forall o_1 \in OUTOBJ_{td}^{MO_1} : \exists o_2 \in OUTOBJ_{td}^{MO_2} : \\ \forall p_i \in \langle [s_p, e_p], l_p \rangle^{o_1} : \exists q_i \in \langle [s_q, e_q], l_q \rangle^{o_2} : \\ (temporal_predicate(p_i, q_i)) \end{aligned} \quad (4.12)$$

The temporal predicate in equation 4.12 considers the period elements p_i and q_i . This means that we use the same period element from both output objects. It is important to note, that the period elements must have the same temporal domain. It is not possible that one period is defined e.g. over time stamps and the other over integers.

Constraints are defined for the start and end points of the period elements. There are no conditions within the period elements. The start and end points of a period are *synchronization points*. In this way we define a *discrete synchronization*. A *continuous synchronization* cannot be defined directly. It is possible to define very small period elements. So, a continuous synchronization can be simulated.

For our movie example the synchronization between two sound sources sometimes must be fine granular. Especially for music a very fine granular synchronization is necessary because even small breaks or delays interfere with the sound impression. Thus, a rough synchronization which takes only the start and end points into account is not enough. The following output condition is suitable for our example. This condition is defined for each tuple of the relation *MOVIE*. The periods are defined for the whole output objects of the sound sources. With the temporal predicate *equal* we define that the period elements p_i and q_i start and end at the same time.

$$\begin{aligned} \forall u \in MOVIE : \\ (\forall o_1 \in \{o \mid o = f_{td}(mo) \wedge mo \in \{u.SOUND_SOURCE_1 \mid movie(u)\}\} : \\ \exists o_2 \in \{o \mid o = f_{td}(mo) \wedge mo \in \{u.SOUND_SOURCE_2 \mid movie(u)\}\} : \\ \forall p_i \in \langle [1, \infty], 1 \rangle^{o_1} : \exists q_i \in \langle [1, \infty], 1 \rangle^{o_2} : \\ p_i \text{ equal } q_i) \end{aligned}$$

Spatiotemporal Conditions define a spatial order between output objects for a certain time. This is only possible if an output object is both time dependent object and spatial output object. A video is an example for such an output object. It has an internal time, therefrom it is time dependent. Of course, a video needs a frame for displaying. Thus, it is also a spatial output object. Another example can be found in our wave field synthesis scenario. There we have sound sources which are points in the listening room. A sound source is represented by an audio and therefore it has an internal time. Hence, we can regard sound sources as spatiotemporal output objects. If we want to define a spatiotemporal output condition we need at least one output object that is a time dependent and spatial output object. The other output object must be a spatial or spatiotemporal output object.

Equation 4.13 shows the formal definition of spatiotemporal output conditions. The output object o_1 is a spatiotemporal output object (e.g. a video). The output object o_2 is a spatial output object (e.g. an image) or also a spatiotemporal output object. Equation 4.13 defines that during the time interval $[s, e]$ specified on o_1 the predicate must be fulfilled. As predicate we use spatial predicates. With other words we can say that the output objects o_1 and o_2 are a group for a certain time. Thus, spatiotemporal output conditions are time restricted.

$$\begin{aligned} \forall o_1 \in OUTOBJ_{f_{td/s}}^{MO_1} : \exists o_2 \in OUTOBJ_{f_s \text{ or } f_{td/s}}^{MO_2} : \\ \forall t \in [s, e]^{o_1} : spatial_predicate(o_1, o_2) \end{aligned} \quad (4.13)$$

As an example we want to define an spatiotemporal output condition according to figure 4.4, but now the output condition has a temporal restriction. The following definition shows the appropriate output condition:

$$\begin{aligned} \forall o_1 \in \{o \mid o = f_{td/s}(mo) \wedge \\ mo \in \{u.SOUND_SOURCE_1 \mid movie(u) \wedge SCENE_NR=1\}\} : \\ \exists o_2 \in \{o \mid o = f_{td/s}(mo) \wedge \\ mo \in \{u.SOUND_SOURCE_1 \mid movie(u) \wedge SCENE_NR=1\}\} : \\ \forall t \in [1, 10]^{o_1} : o_1 \text{ west}(4, 0) o_2 \end{aligned}$$

Both output objects (sound sources) have temporal dimensions as well as spatial dimensions. Therefore, we can use them in a spatiotemporal output condition. We define that during the time units 1 to 10 of output object o_1 this output object is exactly 4 space units west of output object o_2 .

4.7 Notation of Output Constraints

To bring output constraints into a database system, we must define output constraints in a data definition language. This section makes a proposal what such a definition for output constraints could look like.

Our Proposal of a DDL Construct for defining output constraints is shown in figure 4.6. It is an enlargement of the ASSERTION-Definition known from SQL:99.

However, we have to determine whether the proposed specification language (section 4.6) can be mapped to the proposed DDL construct. Elements of the specification language are *sets of output objects*, *temporal and spatial predicates*, *output functions*, and *periods*. Actually, our specification language must be mapped to that part of the DDL that defines the output condition. As the example below shows, SQL is used for defining the output condition. If we introduced output functions and temporal/spatial predicates into SQL, sets of output objects could be generated by select queries. Some output conditions deal with periods (e.g. equation 4.8). These periods are usually defined on the media internal time of time dependent output objects. SQL:99 supports *interval*-types which can be used for defining these periods.

```

CREATE OUTPUT_CONSTRAINT name
  CHECK          output-condition
  REASON         checking-reason [,checking-reason]
  [LENGTH       length-condition [,length-condition] ]
  [REACTION MODIFICATION          action ]
  [REACTION OUTPUT              action ]

```

Figure 4.6: Definition of Output Constraints

For example we want to define an output condition for the *equal* relation shown in figure 4.5. We use the relation *MOVIE* for this example and define the output constraint shown in figure 4.7.

```

CREATE OUTPUT_CONSTRAINT outEqual
  CHECK    not exists (select * from movie
                        where scene_nr=1 and
                        not (play(sound_source_1) equal play(sound_source_2)))
  REASON   modification
  LENGTH   play(sound_source_1).setLength(select sound_source_1.getLength()
                                             from movie where scene_nr=1)
              play(sound_source_2).setLength(select sound_source_2.getLength()
                                             from movie where scene_nr=1)
  REACTION MODIFICATION      notification

```

Figure 4.7: DDL Construct for Output Constraint

The output constraint defined above has the name *outEqual*. We use an output function called *play()* in the output condition. This function takes stored sound sources and builds output objects from them. Furthermore, this functions plays the output objects during the data output. In the output condition defined in the **CHECK**-clause we define that no output object from *sound_source_1* exists in the first scene that cannot be played equally to the output object of *sound_source_2*. In the **REASON**-clause it is defined that the output condition must be checked after modifications on the media data. When the output condition is violated the system should notify the user. For checking output conditions the temporal length of the output objects is needed. In the **LENGTH**-clause we detect the duration of the required output objects with a **SELECT**-statement and define this duration as length for the output objects used in the output constraint. It is assumed that the sound source attributes have a complex data type that provides the method *getLength()* to determine the temporal length of an audio.

Actually, in this example no **LENGTH**-definition is needed because the length is directly determined by the required audio data. Usually, no explicit **LENGTH**-definition is done. However, the **LENGTH**-clause can be used for a hard length definition of an output object (e.g. *output_object.setLength(10)*).

An XML Based Approach for defining output constraints have been developed as well. The advantage of this approach is its easy implementation because we can use existing utilities for parsing and checking the XML syntax. The XML structure below also defines the output constraint *outEqual*. All parts of the above DDL construct can be found in the XML structure as well. The output objects *o1* and *o2* are defined by selecting the attributes *sound_source_1* and *sound_source_2* and applying the output function *play* on them. The conditions-clause specifies that both output objects must be output equally. Furthermore, the temporal length of both output objects is defined. The rest of the XML definition deals with the reason for constraint checking and the reaction in case of constraint violations.

```

< OutputObject >
  < Name > o1 < /Name >
  < Value > select play(sound_source_1) from movie where scene_nr=1 < /Value >
< /OutputObject >
< OutputObject >
  < Name > o2 < /Name >
  < Value > select play(sound_source_2) from movie where scene_nr=1 < /Value >
< /OutputObject >
< Conditions >
  < Constraint name = "equals" >
    < Object > o1 < /Object >
    < Object > o2 < /Object >
  < /Constraint >
  < Constraint name = "length" >
    < Object > o1 < /Object >
    < Parameter > select sound_source_1.getLength() from movie where scene_nr=1
    < /Parameter >
  < /Constraint >
  < Constraint name = "length" >
    < Object > o2 < /Object >
    < Parameter > select sound_source_2.getLength() from movie where scene_nr=1
    < /Parameter >
  < /Constraint >
< /Conditions >
< Reason > modification < /Reason >
< ReactionModification > notification < /ReactionModification >

```

4.8 Conclusion

The contribution of this chapter was the developing of formal specifications of output constraints. The most important points from this chapter are the followings:

- A taxonomy for output constraints was built. In this work we use a classification of output constraints which builds constraint classes according to the restricted

output feature. The main constraint groups in this classification are output constraints on output parameters and output constraints for synchronization.

- Each class of output constraints needs a formal specification. Constraints on output parameters can be defined very easily by using normal inequalities. However, output constraints for synchronizations need special predicates. We use Allen-Relations for defining temporal relationships between output objects. Allen-Relations must be enlarged for supporting quantitative relationships. Therefore, we have introduced restricted Allen-Relations. We have seen that Allen-Relations can also be applied on spatial synchronization constraints. Thus, we got an unified method for defining temporal and spatial output constraints.
- We have proposed a formal constraint language for output constraints. This language is based on first order logic and temporal logic. Beside the formal specification we have proposed a DDL specification and an XML structure which can be used in practice for defining output constraints.

Chapter 5

Internal Representation of Output Constraints

For checking output constraint an adequate representation of output constraints inside the database system is required (figure 5.1). The contribution of this chapter is developing this database internal representation of output constraints.

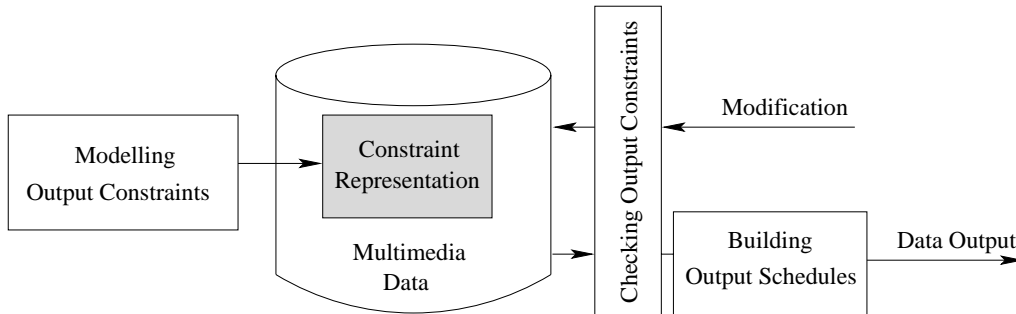


Figure 5.1: Processing of Output Constraints

By considering all aspects of managing output constraints in detail we will deduce requirements on the database internal representation of output constraints. Based on these requirements we will propose a database internal representation of output constraints. Furthermore, we will develop a transformation of our constraint specification language into the database internal representation.

5.1 Process of Output Constraint Management

Output constraints must be taken into account during *modelling*, *data modifications*, *data output* and for *data organization*. Figure 5.2 shows general process of the handling output constraints. The boxes are new components which have to be integrated into a multimedia database system, the arrows describe data flows.

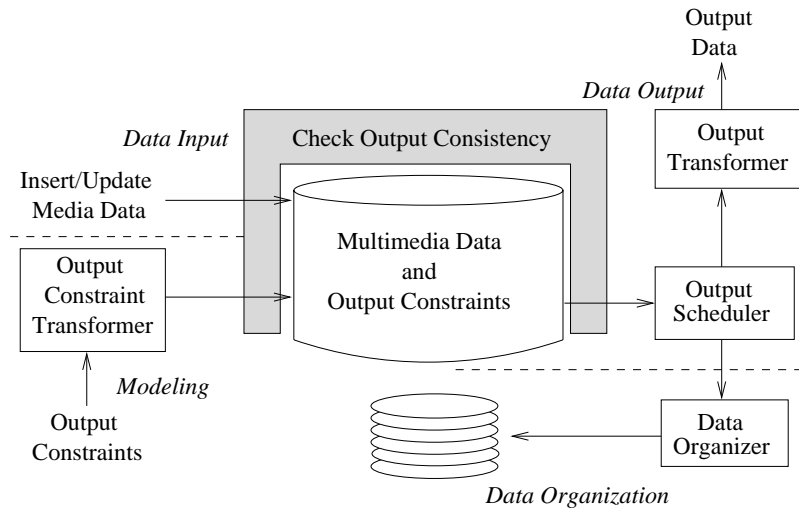


Figure 5.2: Process of Using Output Constraints

The Modelling should be supported by an authoring tool. Defining output constraints must be very intuitive for the author. Usually, the authoring tool is a stand alone software. This means, it is not integrated into the multimedia database system, but the database system must be able to handle the results of the modelling process. As mentioned before the result of the modelling process can be stored as an XML-file. The database system must be able to import output constraints from various sources.

A transformer pre-processes the output constraints and stores them conveniently into the database system. The internal representation of the output constraint must allow an efficient constraint check. Sometimes the media data required by the output constraints are already stored in the database system. In this case the transformer must check whether the media data and output constraints are suitable or not. The transformer must also check the output constraint sets for inconsistencies.

If we want to change output constraints, we cannot do this directly in the database system. The transformer must extract the output constraints from the database system, then they can be modified by the authoring tool.

Data Modifications means update or insert operation on media data. Usually, media data like video data or audio data are edited with special software which is not part of the database system. These programs do not have knowledge about the output constraints. They produce media data (e.g. audio data or video data) without any consideration of output constraints.

The media data produced by the editing software must be inserted into the database system. During this data input process output constraints must be checked to avoid inconsistencies of media data. Only if we check the output consistency on this point, it can be guaranteed that an adequate data output is possible. It is important to note that this check does not guarantee the output consistency for a certain output process. We

need an efficient consistency check, otherwise a delay occurs during insert and update operations.

The Data Output of media data must be compatible to the output constraints. Thus, an output order of output objects is needed that fulfills all output constraints. The *scheduler* is the component that builds this orders, also named *output schedules*. When temporal and spatial output constraints are defined, we have to produce output schedules for the temporal and spatial output order as well.

A format independent storage of media data in a multimedia database system is desired. Therefore, format transformation during the output process is necessary, that is realized by the *output transformer*. Usually, the user wants to get a data stream, but it is also possible that the transformer builds a SMIL-script with the output schedule. In this case the database system replies the database query with a SMIL-script and stores the required media data on the file system.

If we replay a database query with a multimedia data stream, we have to check the output consistency during the output process. Mainly, this means that constraints on output parameters must be checked permanently. There are two reasons for violating this kind of output constraints. The first arises from a lack of resources. As an example if the required bandwidth is not available, it is not possible to guarantee an appropriate frame rate for a video. Requests to the database system are another chance to violate output constraints. A user can define a certain frame rate for a video in a database query that is not allowed from the data designer. Thus, database queries must be checked considering output constraints.

The Data Organization must take the output schedule into account. The temporal output schedule defines exactly which data we need at which point in time. For output media data we need time for several operations. As an example we need some time for searching the required media data. The time we need for reading data from hard disk and time for transforming media data must also be taken into account.

The data organizer has some options to reduce the time required for data output. As an example it can create useful index structures to find the required media data faster. Partitioning is another possibility. We can increase the delivery rate if we store media data on several hard disks. As an example if we want to output two audios equally, it is useful to store them on different hard disks. Caching is also a way to make the data output faster. Sometimes an audio is needed twice. In this case it could be useful to cache that audio.

Requirements of a Database Internal Representation of Output Constraints

Considering the above process of handling output constraints we make the following requirements on a database internal representation of output constraints:

- There must be a possibility to produce the database internal representation based on the constraint language which is used for modelling output constraints.
- Each class of output constraints needs a database internal representation.
- The database internal representation must allow efficient constraint checking in case of manipulations on media data or constraints.
- An efficient check of output constraints during the data output process must be supported.
- It must be possible to produce temporal and spatial output schedules based on output constraints.

5.2 Representation of Output Constraints

This section shows how a database internal representation of output constraints can be built for the several output constraint classes shown in chapter 4. To allow an efficient checking of output constraints the predicates used in the output constraint must be organized in such a way that they can be evaluated easily. The main idea is transforming output constraints into inequalities. These inequalities can be used for building a graph representation of output constraints which allows an efficient consistency check.

5.2.1 Output Constraints on Output Parameters

As we know, these classes of output constraints restrict specific output parameters which are properties of media object. The resolution of images or the frame rate of a video are examples of output parameters. Usually, these parameters are stored as metadata, belonging to media objects. Thus, the database internal representation is equal to the definition of the output constraint. Take a look on the example used in section 4:

$$\forall o \in \{o \mid o = f_s(mo) \wedge mo \in \{z.MOVIE_SCENE \mid movie(z)\}\} : \\ hor_resolution(o) \geq 1365$$

The output constraint above restricts the horizontal resolution of a video. Resolution is a typical example of metadata, belonging to a video. Therefore, this output constraint can be transformed in a constraint $hor_resolution(o) \geq 1365$ on the according metadata. It is important to note that this constraint is no integrity constraint, it only restricts the resolution during data output.

If we deal with periodic output constraints, the period is not of interest for the database internal constraint representation. The period has only impact on the point in time of checking output constraints. Thus, we need an adequate representation of the period for example in the catalogue of the database system.

5.2.2 Output Constraints for Synchronization

This class of output constraints can be divided into two groups: one group uses temporal predicates and the other group uses spatial predicates.

Temporal Output Constraints

We used Allen-Relations in our constraint language. However, these relations are unsuitable if we want to check consistency, since checking consistency of a set of Allen-Relations is NP-hard. Thus, we use *Difference Constraints* as database internal representation for temporal predicates.

Difference Constraints

Difference constraints are a specific class of real valued linear constraints. While generalized linear constraints have the form

$$a_1x_1 + a_2x_2 + \dots + a_nx_n \leq c$$

where a_1, \dots, a_n, c are *relational numbers*, and x_1, \dots, x_n range over the *real numbers*, difference constraints have the form

$$x_1 - x_2 \leq c.$$

Thus, difference constraints are a special case of linear constraints where:

- There are only two variables (x_1 and x_2) and the constant c .
- One variable has coefficient 1 while the other has coefficient -1.

Due to the fact that difference constraints have a very tightly restricted syntactic form, it turns out that they are very easy to solve. A set of difference constraints can be solved in polynomial time [CLR00].

Actually, some authors [CPS96, CRS98] use different constraints to model temporal or spatial relationships between media objects directly. The idea behind this approach is to use start and end points of temporal or spatial intervals as variables for the difference constraints.

How to use difference constraints can easily be exemplified on our movie example. It is assumed that for a certain scene *sound source 1* should play as long as *sound source 2*. By using difference constraints this can be expressed as follows:

$$\begin{array}{ll} start(SOUND_SOURCE_1) - start(SOUND_SOURCE_2) & \leq 0 \\ start(SOUND_SOURCE_2) - start(SOUND_SOURCE_1) & \leq 0 \\ end(SOUND_SOURCE_1) - end(SOUND_SOURCE_2) & \leq 0 \\ end(SOUND_SOURCE_2) - end(SOUND_SOURCE_1) & \leq 0 \end{array}$$

SOUND_SOURCE_1 and *SOUND_SOURCE_2* are used here as temporal intervals that represent temporal lengths of audio data. With *start* and *end* the start and end points of these intervals are specified. The above example can be expressed by means of

Allen-Relations as *SOUND_SOURCE_1 equal SOUND_SOURCE_2*. It is easy to see that Allen-Relations can be handled by the user much more easily as difference constraints. A small set of Allen-Relations can lead to a very large and confusing amount of difference constraints. Thus, we cannot confront the database manager or the producer of the media data directly with such an unmanageable constraint set.

Transformation of Allen-Relations

To produce difference constraints from Allen-Relations a suitable transformation is needed. The basic idea behind this transformation uses the fact that each temporal interval involved in an Allen-Relation has a start and an end point. If we use Allen-Relations, exact time points for intervals are never defined, only the interval length is considered. Thus, start and end point of temporal intervals are variable. Therefore, they can be used as variables in difference constraints.

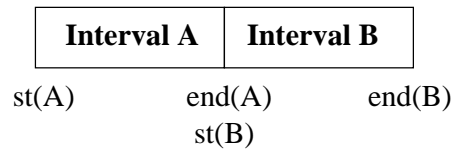


Figure 5.3: Allen-Relation *A meets B*

For example we want to demonstrate the transformation of the Allen-Relation *A meets B* into a set of difference constraints. Figure 5.3 shows the intervals A and B which are involved in the relation *A meets B*. The characteristics of this Allen-Relation is that the start point of interval B is at the same time as the end point of interval A, formally this can be expressed as $end(A) - st(B) = 0$. Because of the fact that difference constraints always have the form $x_1 - x_2 \leq c$ a set of inequalities: $(end(A) - st(B) \leq 0) \wedge (st(B) - end(A) \leq 0)$ must be used.

For each Allen-Relation a corresponding set of difference constraints can be found. Table 5.1 shows our transformation of restricted Allen-Relations into difference constraints. Because of its strict form, $x_1 - x_2 \leq c$, a constant c must be specified in difference constraints. If we transform Allen-Relations into difference constraints, this constant must originally come from Allen-Relations.

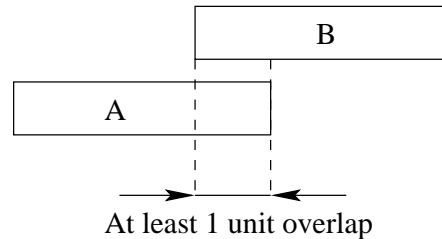


Figure 5.4: Allen-Relation *A overlaps B*

In the example mentioned above we had the simple case that this constant was 0. This is not a general fact but rather a special characteristic of some Allen-Relations (meets, finishes, starts, equals). The modifications we made on Allen-Relations (table 4.2) allow quantitative definitions. The concrete quantitative values can be used as constants for difference constraints. Take the restricted Allen-Relation $A \text{ before}(c_1) B$ as example. The constant c_1 defines the minimum distance between the end of interval A and the start of interval B. So, that constant can directly be used for the appropriate difference constraint $end(A) - st(B) \leq -c_1$.

Allen-Relation	Difference Constraints
$A \text{ meets } B$	$(end(A) - st(B) \leq 0) \wedge (st(B) - end(A) \leq 0)$
$A \text{ finishes } B$	$(end(A) - end(B) \leq 0) \wedge (end(B) - end(A) \leq 0)$
$A \text{ starts } B$	$(st(A) - st(B) \leq 0) \wedge (st(B) - st(A) \leq 0)$
$A \text{ equals } B$	$(end(A) - end(B) \leq 0) \wedge (end(B) - end(A) \leq 0)$ $\wedge (st(A) - st(B) \leq 0) \wedge (st(B) - st(A) \leq 0)$
$A \text{ before}(c_1) B$	$(end(A) - st(B) \leq -c_1)$
$A \text{ before}_V(c_1, l) B$	$(end(A) - st(B) \leq -c_1) \wedge$ $(st(B) - end(A) \leq c_1 + l)$
$A \text{ overlaps}(c_1, c_2) B$	$(st(A) - st(B) \leq -c_1) \wedge (end(A) - end(B) \leq -c_2)$ $\wedge (st(B) - end(A) \leq -1)$
$A \text{ overlaps}_{V_s}(c_1, l, c_2) B$	$(st(A) - st(B) \leq -c_1) \wedge (st(B) - st(A) \leq (c_1 + l))$ $\wedge (end(A) - end(B) \leq -c_2)$ $\wedge (st(B) - end(A) \leq -1)$
$A \text{ overlaps}_{V_e}(c_1, c_2, l) B$	$(st(A) - st(B) \leq -c_1) \wedge (end(B) - end(A)$ $\leq (c_2 + l)) \wedge (end(A) - end(B) \leq -c_2)$ $\wedge (st(B) - end(A) \leq -1)$
$A \text{ overlaps}_{V_{se}}(c_1, l, c_2, m) B$	$(st(A) - st(B) \leq -c_1) \wedge (st(B) - st(A) \leq (c_1 + l))$ $\wedge (end(B) - end(A) \leq (c_2 + m)) \wedge$ $(end(A) - end(B) \leq -c_2) \wedge (st(B) - end(A) \leq -1)$
$A \text{ during}(c_1, c_2) B$	$(st(B) - st(A) \leq -c_1)$ $\wedge (end(A) - end(B) \leq -c_2)$
$A \text{ during}_{V_s}(c_1, l, c_2) B$	$(st(B) - st(A) \leq -c_1) \wedge (st(A) - st(B) \leq (c_1 + l))$ $\wedge (end(A) - end(B) \leq -c_2)$
$A \text{ during}_{V_e}(c_1, c_2, l) B$	$(st(B) - st(A) \leq -c_1) \wedge (end(A) - end(B) \leq$ $-c_2) \wedge (end(B) - end(A) \leq (c_2 + l))$
$A \text{ during}_{V_{se}}(c_1, l, c_2, m) B$	$(st(B) - st(A) \leq -c_1) \wedge (st(A) - st(B) \leq (c_1 + l))$ $\wedge (end(A) - end(B) \leq -c_2) \wedge$ $(end(B) - end(A) \leq (c_2 + m))$

Table 5.1: Transforming Allen-Relations into Difference Constraints

Generally, the transformation of Allen-Relations into difference constraints is very intuitive. Only the *overlaps* relation has an anomaly. Considering figure 5.4 we see that it is necessary to define a difference constraint that guarantees a real overlap between

the intervals. Therefore, we introduce the difference constraint $st(B) - end(A) \leq -1$ for the *overlaps* relations.

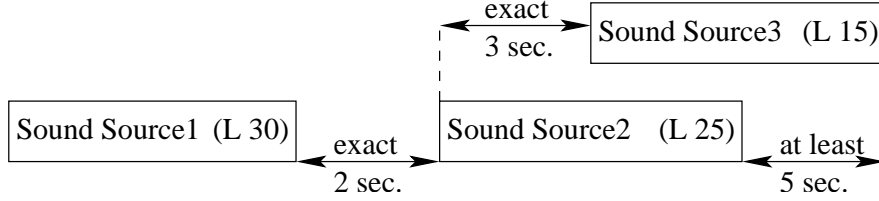


Figure 5.5: Temporal Conditions for a Sound Scene

To exemplify the transformation of Allen-Relations we want to use our movie example. Figure 5.5 shows the relationships of sound sources in a sound scene that must be modelled. The length of the intervals is denoted as $(L x)$. For example, we can see from $(L 30)$ in the interval of *Sound Source1* that it has a duration of 30 time units. The following temporal output condition can be used for expressing the scene in our output constraint language:

$$\begin{aligned}
 &\forall o_1 \in \{o \mid o = f_{td}(mo) \wedge \\
 &\quad mo \in \{u.SOUND_SOURCE_1 \mid movie(u) \wedge SCENE_NR='1'\}\} : \\
 &\quad \exists o_2 \in \{o \mid o = f_{td}(mo) \wedge \\
 &\quad \quad mo \in \{u.SOUND_SOURCE_2 \mid movie(u) \wedge SCENE_NR='1'\}\} : \\
 &\quad \exists o_3 \in \{o \mid o = f_{td}(mo) \wedge \\
 &\quad \quad mo \in \{u.SOUND_SOURCE_3 \mid movie(u) \wedge SCENE_NR='1'\}\} : \\
 &\quad (o_1 \text{ before}_V(2, 0) o_2) \wedge (o_2 \text{ overlaps}_{V_s}(3, 0, 5) o_3)
 \end{aligned}$$

It is easy to see that this output condition handles output objects which are produced from the audios stored as sound sources for the scene '1' in the table *MOVIE*. The following set of difference constraints are produced when we transform above temporal output condition:

Temporal Condition	Difference Constraints
$o_1 \text{ before}_V(2, 0) o_2$	$(st(o_2) - end(o_1) \leq 2 + 0) \wedge (end(o_1) - st(o_2) \leq -2)$
$o_2 \text{ overlaps}_{V_s}(3, 0, 5) o_3$	$(st(o_3) - st(o_2) \leq 3 + 0) \wedge (st(o_2) - st(o_3) \leq -3) \wedge$ $(end(o_2) - end(o_3) \leq -5) \wedge (st(o_3) - st(o_2) \leq -1)$

Table 5.2: Transformed Temporal Output Condition

Spatial Output Constraints

For defining spatial output constraints on salient objects we use the MBR's of these objects. As already mentioned we make projections of the MBR's on each spatial dimension and use Allen-Relations to define the spatial constraints for each spatial

dimension separately. Spatial output constraints on salient objects can be transformed into difference constraints according to table 5.1.

Spatial relationships between point objects can be defined in the constraint language by using basic spatial predicates like *west*, *east*, *south*, *north*. We assume that each spatial point is 2-dimensional. We use the Euclidean metric and the Cartesian coordinate system. Each spatial point has an *X* and a *Y Coordinate*. Therefore, the transformation into difference constraints is obvious, we can see it in table 5.3.

Spatial-Relation	Difference Constraints
<i>pointA west(c,l) pointB</i>	$(X_A - X_B \leq -c) \wedge (X_B - X_A \leq (c + l))$
<i>pointA east(c,l) pointB</i>	$(X_A - X_B \leq (c + l)) \wedge (X_B - X_A \leq -c)$
<i>pointA south(c,l) pointB</i>	$(Y_A - Y_B \leq -c) \wedge (Y_B - Y_A \leq (c + l))$
<i>pointA north(c,l) pointB</i>	$(Y_A - Y_B \leq (c + l)) \wedge (Y_B - Y_A \leq -c)$

Table 5.3: Transforming Spatial Relations for Point Objects into Difference Constraints

To give an example we use the following spatial output condition from section 4.3.4:

$$\begin{aligned}
 &\forall o_1 \in \{o \mid o = f_s(mo) \wedge \\
 &\quad mo \in \{u.SOUND_SOURCE_1 \mid movie(u) \wedge SCENE_NR = 'I'\}\} : \\
 &\quad \exists o_2 \in \{o \mid o = f_s(mo) \wedge \\
 &\quad \quad mo \in \{u.SOUND_SOURCE_2 \mid movie(u) \wedge SCENE_NR = 'I'\}\} : \\
 &\quad (o_1 \text{ west}(4.0) o_2)
 \end{aligned}$$

The transformation produces the following set of difference constraints:

Spatial Condition	Difference Constraints
$o_1 \text{ west}(4.0) o_2$	$(X_{o_1} - X_{o_2} \leq -4) \wedge (X_{o_2} - X_{o_1} \leq -4 + 0)$

Table 5.4: Transformed Spatial Output Condition

5.2.3 Transforming Temporal and Spatial Lengths

Allen-Relations define relationships between intervals. Up to now we have only considered those predicates which describe relationships. For evaluating temporal or spatial output constraints the lengths of the involved intervals must be taken into account. This means an adequate representation of temporal and spatial interval lengths is necessary (table 5.5). For achieving an uniform representation for all constraints we also use difference constraints for these length constraints.

Considering the sound scene of figure 5.5 we have the intervals for *Sound Source1*, *Sound Source2* and *Sound Source3* with the lengths 10, 15 and 25. We can transform these lengths conditions into difference constraints as follows:

Temporal / Spatial Lengths	Difference Constraints
$A \text{ length}(c)$	$(end(A) - st(A) \leq c) \wedge (st(A) - end(A) \leq -c)$

Table 5.5: Transforming Temporal and Spatial Lengths

Temporal Lengths	Difference Constraints
$o_1 \text{ length}(30)$	$(end(o_1) - st(o_1) \leq 30 \wedge (st(o_1) - end(o_1) \leq -30))$
$o_2 \text{ length}(15)$	$(end(o_2) - st(o_2) \leq 15 \wedge (st(o_2) - end(o_2) \leq -15))$
$o_3 \text{ length}(25)$	$(end(o_3) - st(o_3) \leq 25 \wedge (st(o_3) - end(o_3) \leq -25))$

Table 5.6: Lengths Constraints for Figure 5.5

5.3 Storage Structure of Output Constraints

Traditional integrity constraints are managed by a database system by using special system tables. For handling output constraints we have introduced special management tables as well. These tables store different constraints produced by output conditions as well as information required for checking consistency. In this section we will only introduce the table that manages difference constraints:

Constraint_Graphs(ID, *start_node*, *end_node*, *weight*, *C_ID*)

The attribute *ID* is an identifier of difference constraints. The attribute *start_node* stores the second variable of a difference constraint while the first variable is stored as *end_node*. The constant *c* used in difference constraints is stored as *weight*.

The attribute *C_ID* stores an identification for each restricted Allen-Relation. This is necessary because a set of difference constraints belongs to an Allen-Relation. In other words, all difference constraints produced by a specific Allen-Relation have the same *C_ID* value. We need this information for reconstructing the original restricted Allen-Relations on the basis of these difference constraints. Furthermore, in case of removing Allen-Relations, we have to delete all difference constraints belong to these Allen-Relations.

The table entries of table *Constraint_Graphs* are used for building a *constraint graph* (chapter 6) as main memory data structure which is required for checking output consistency.

As an example we want to show the table entries for the restricted Allen-Relation $o_1 \text{ before}_v(2, 0) o_2$ used in table 5.2. The difference constraints produced from this Allen-Relation are:

$$end(o_1) - st(o_2) \leq -2 \qquad st(o_1) - end(o_2) \leq 2$$

The table entries which are inserted in the *Constraint_Graph* table for these difference

constraints are shown in table 5.7. It is easy to see that the table entries are equal to the

<u>ID</u>	start_node	end_node	weight	C_ID
1	$st(o_2)$	$end(o_1)$	-2	c1
2	$end(o_2)$	$st(o_1)$	2	c1

Table 5.7: Table Entries in Relation *Constraint_Graph*

defined difference constraints. It is important to note that both entries have the same *C_ID* value. In our example this value is *c1*. This is because both difference constraints come from the restricted Allen-Relation $o_1 \text{ before}_v(2, 0) o_2$.

5.4 Conclusion

The intention of this chapter was to propose a representation for output constraints which allows an efficient constraint checking. We have seen that output constraints on output parameters can easily be transformed into constraints on metadata.

Allen-Relations are used in the constraint language for defining output constraint for synchronization. The method of transforming Allen-Relations into a database internal representation requires only two steps. First, Allen-Relations must be transformed into difference constraints. This transformation is shown in table 5.1 and table 5.3. In a second step difference constraints must be stored in a special management table which we have introduced in section 5.3.

Chapter 6

Output Constraints and Data Consistency

Checking output constraints is required during data input as well as during data output (figure 6.1). The contribution of this chapter is proposing concepts for checking output constraints. Especially, in case of modifications of stored media data as well as in case of modifications of output constraints we should check output consistency. Otherwise there is a risk that stored media cannot be output in the required manner.

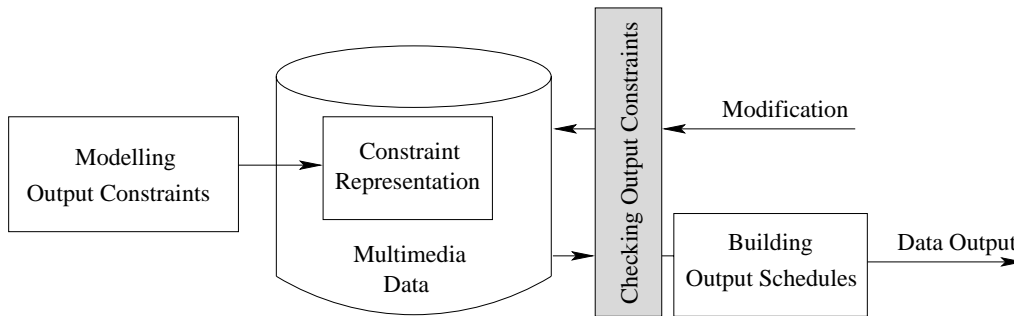


Figure 6.1: Processing of Output Constraints

Especially, output objects involved in output constraints for synchronization should be checked on output consistency. We can detect by these checks whether the temporal and spatial lengths of the media data are adequate to the defined synchronization constraints. This is a necessary precondition to fulfill these output constraints during the data output.

This chapter proposes two approaches for checking synchronization output constraints after database operations like *insert*, *update* and *delete*. Because of the fact that the consistency check is part of the data modification transaction, the execution of these consistency checks must be very fast. Therefore, some optimizations for consistency check are proposed in this chapter as well.

6.1 Execution Model of Output Constraints

If we integrate output constraints into a multimedia database system, we have to define the processing of output constraints. Especially, we must integrate output constraints into query processing. Execution models for integrity constraints and triggers are known from SQL:99 [CPM96]. Therefore, we propose in figure 6.2 a similar execution model for output constraints.

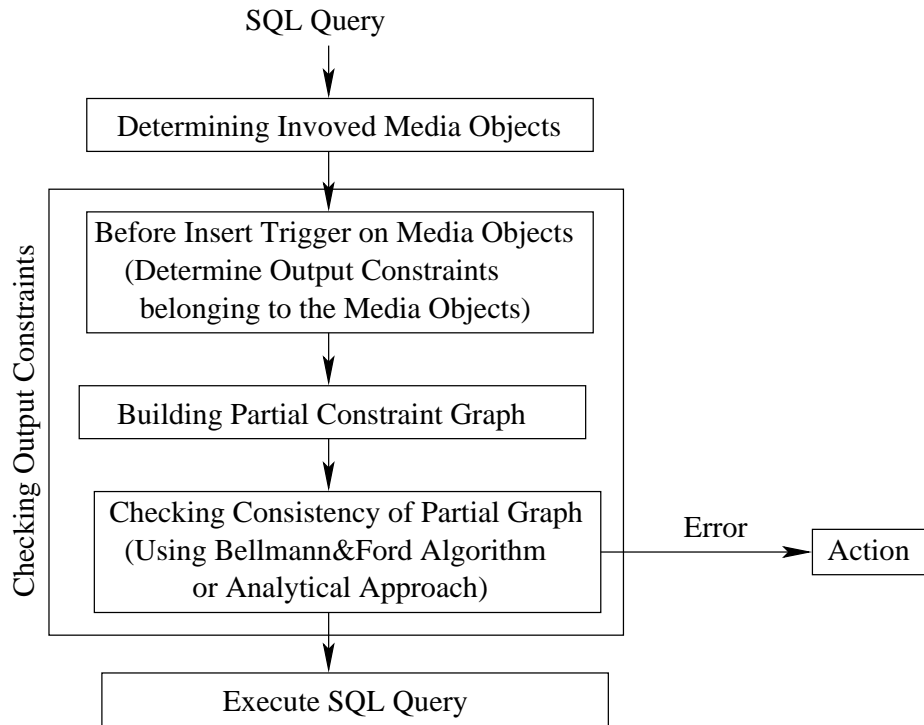


Figure 6.2: Execution Model for Output Constraints

First, we must parse the SQL query for determining the involved media objects. In a next step output constraints must be checked. Thus, a before insert trigger is build on media objects. This trigger checks whether output constraints exist affecting the media object. In practice this can be determined by checking the table *Constraint_Graph* of entries concerning the media object.

Usually a graph representation of output constraints is used for managing output constraints in main memory. However, this constraint graph potentially includes many output constraints. Therefore, we build a partial graph that only includes the output constraints we have to check.

The next step is to check the consistency of this partial graph. For checking consistency we can use either a graph theoretical approach which is based on the *Bellmann and Ford Algorithm* (section 6.2) or an analytical approach (section 6.4).

If the consistency check determines no inconsistencies the SQL query will be ex-

ecuted, otherwise the reaction specified in the output constraint definition will be executed.

6.2 Consistency Check using Graph Theoretical Approach

As mentioned above a graph theoretical approach can be used for checking consistency of sets of output constraints. Actually, we check the consistency of difference constraint sets since we have transformed output constraints from its specification language into a database internal representation.

Consistency of a Set of Difference Constraints

Now we want to check consistency of sets of difference constraints. One possibility to check consistency is to solve a set of difference constraints. More precisely, solution means computing a concrete number for each start and end point of an interval used in a difference constraint. If there exists no solution, the set of difference constraints is inconsistent.

If we want to use a graph theoretical approach for solving a set of difference constraints, we first must build a constraint graph based on difference constraints. Thus, each variable of the constraint set becomes a node in a graph and each inequality becomes a directed edge with the weight of the constant c . It is a known fact in graph theory that a set of difference constraints is solvable if a shortest paths can be found from a start node to all the other nodes of the graph [CRS98].

Different algorithms are known to determine shortest paths in a graph. We use the well known algorithm of *Bellmann and Ford* [CLR00, RSJM99]. This algorithm is suitable for our problem because it can deal with negative edge weights and its complexity is $O(n \cdot m)$, where m is the number of edges and n is the number of nodes in a graph. Applying to a set of difference constraints the complexity of *Bellmann and Ford* depends on the number of output objects involved in this set of difference constraints and on the number of difference constraints contained in the set.

The constraint graph on which the *Bellmann and Ford* algorithm works must be built according to the following definitions.

Definition 7 ($\langle V, C \rangle$). is a *system of Difference Constraints*. V is a set of variables and C is a set of linear inequalities of the form:

$$v_i - v_j \leq c_k \text{ with } v_i, v_j \in V, c_k = \text{constant}, 1 \leq i, j \leq n, 1 \leq k \leq m$$

So, the system has m linear inequalities and n variables.

Definition 8 ($G = \langle V, E \rangle$). is a directed and weighted *constraint graph*. V is the set of nodes and E is the set of weighted edges defined as follows:

$$V = \{v_0, v_1, \dots, v_n\}$$

$$E = \{(v_j, v_i) : v_i - v_j \leq c_k, 1 \leq i, j \leq n, 1 \leq k \leq m\} \cup \{(v_0, v_1), (v_0, v_2), \dots, (v_0, v_n)\}$$

$$\begin{aligned} \text{edge weight: } w(v_j, v_i) &= c_k, j > 0 \\ w(v_0, v_n) &= 0, n > 0 \end{aligned}$$

It is important to note that the node v_0 is introduced. Actually, the *Bellmann and Ford* algorithm must take each node and determine the shortest paths to all other nodes. To simplify matters, v_0 has an edge with weight 0 to each other node. Thus, only v_0 is necessary as start node for *Bellmann and Ford*.

During computing shortest paths the *Bellmann and Ford* algorithm detects negative cycles. A *negative cycle* has the effect that a path that is already detected as shortest path becomes shorter with each run through the cycle. In other words, no shortest path can be detected. Thus, the set of difference constraints has no solution because it is inconsistent [CLR00]. In this case algorithm yields *FALSE* as return value otherwise it returns *TRUE*.

As an example we will build a constraint graph for our movie example. The required difference constraints have been defined in table 5.6 and table 5.2. Objects o_1, o_2, o_3 are output objects for the stored audio data. The resulting constraint graph $G = \langle V, E \rangle$ is defined as follows:

$$\begin{aligned} V &= \{v_0, st(o_1), end(o_1), st(o_2), end(o_2), st(o_3), end(o_3)\} \\ E &= \{ (st(o_1), end(o_1)), (end(o_1), st(o_1)), (st(o_2), end(o_2)), (end(o_2), st(o_2)), \\ &\quad (st(o_3), end(o_3)), (end(o_3), st(o_3)), (end(o_1), st(o_2)), (end(o_2), st(o_1)) \\ &\quad (st(o_2), st(o_3)), (st(o_3), st(o_2)), (end(o_3), end(o_2)), (st(o_3), end(o_2)) \} \\ &\quad \cup \{(v_0, st(o_1)), (v_0, end(o_1)), (v_0, st(o_2)), (v_0, end(o_2)), \\ &\quad (v_0, st(o_3)), (v_0, end(o_3))\} \end{aligned}$$

edge weights:

$$\begin{aligned} w(st(o_1), end(o_1)) &= 30, w(end(o_1), st(o_1)) = -30, \\ w(st(o_2), end(o_2)) &= 15, w(end(o_2), st(o_2)) = -15, \\ w(st(o_3), end(o_3)) &= 25, w(end(o_3), st(o_3)) = -25, \\ w(end(o_1), st(o_2)) &= 2, w(end(o_2), st(o_1)) = -2, \\ w(st(o_2), st(o_3)) &= 3, w(st(o_3), st(o_2)) = -3, w(end(o_2), st(o_3)) = -1, \\ w(end(o_3), end(o_2)) &= -5, w(v_0, st(o_1)) = 0, \\ w(v_0, end(o_1)) &= 0, w(v_0, st(o_2)) = 0, \\ w(v_0, end(o_2)) &= 0, w(v_0, st(o_3)) = 0, w(v_0, end(o_3)) = 0 \end{aligned}$$

Figure 6.3 shows the corresponding graph. If we apply the *Bellmann and Ford* algorithm on this graph, it returns *TRUE* because there is no negative cycle in the graph.

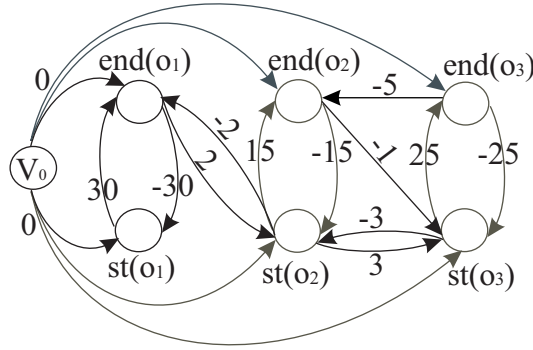


Figure 6.3: Constraint Graph for an Audio Scene

6.3 Building a Partial Constraint Graph

In the following we want to consider optimizations for consistency checking. We assume that in an initial state all stored media data are consistent with regard to the defined output constraints.

Generally, consistency must be checked either after media objects have changed or after output constraints have changed. For optimizations we must consider modifying the lengths of output objects and modifying the output constraints separately. The aim of the optimization is to check only a partial constraint graph instead of the whole graph.

Thus, a partial graph must be generated to which the effects of media or constraint modifications are restricted. A basic approach of producing such a partial graph is to determine the connected graph starting from the modified point in the original constraint graph. This technique often leads to a partial graph that is larger than necessary.

6.3.1 Restrictions on Constraint Graphs

Usually, the presentation of a multimedia document is designed in a structured way. This means, media objects are arranged in a special graph structure. An adequate structure to define presentations of multimedia documents is a tree [BF98, BK99]. Since a tree has no cycles, iterations cannot be defined in multimedia presentations based on a tree structure. Thus, some authors [BK99] have introduced special constructs for iterations into a multimedia presentation language.

In almost the same manner, which is used to define a presentation of a multimedia document, the output condition is defined. Usually, iterations are not important for output constraints. Output constraints only check if output objects can be output in a defined order. How often this output occurs is not of interest. However, it is possible to transform an iteration into a sequence of output objects and relationships. Therefore, from a practical point of view a tree structure is adequate for the definition of conditions in output constraints. Thus, the following restrictions are made:

- There is exactly one start object, called v_0 , it has no predecessor and no dimensional expansion. This means, the output object v_0 has neither a temporal length nor a spatial size.
- With the exception of v_0 each other output object has exactly one output object as predecessor and an arbitrary number of output objects as successors. Furthermore, v_0 is predecessor of each output object.
- Except for v_0 , each output object has at least one dimensional expansion (a temporal or spatial length).

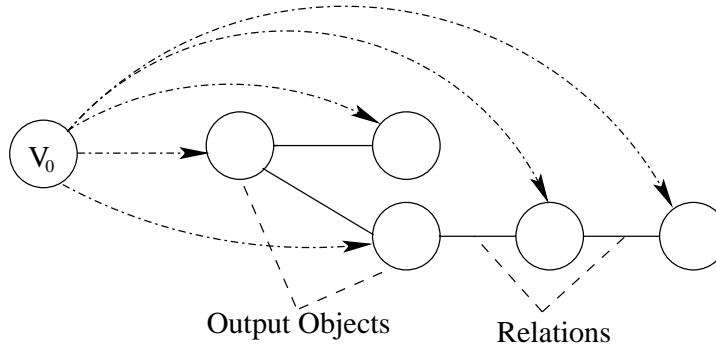


Figure 6.4: Structure of a Constraint Graph

Figure 6.4 shows the general structure of output objects and relations between them. Output objects are depicted as single nodes in figure 6.4. This means the start and end node of an output object are depicted together as one node. It is easy to see that v_0 is the root of the tree. This structure has the following advantages concerning constraint checking:

- Only the node v_0 is start node for the *Bellmann and Ford* algorithm.
- v_0 is only a source for edges. This means, there is no cycle in the graph where v_0 is involved. If we consider the graph in figure 6.4 without v_0 and its edges, it is a tree. Because a tree has no cycles, negative cycles can occur only between two output objects which are neighbours in the tree. Thus, modifications of Allen-Relations or of output objects can only produce a local negative cycle between output objects that are neighbours.

6.3.2 Modifications of Allen-Relations

First we want to consider the optimization potential that comes from Allen-Relations. So, this section only deals with output constraint modifications and not with modifications of output objects. In figure 6.5 we can see dashed edges between output objects

which comes from Allen-Relations. We want to investigate what impact modifications on these edges have on the consistency of the constraint graph.

As mentioned above negative cycles on the shortest path between v_0 and another node can arise from modifications of the relations between output objects. However, not each Allen-Relation can lead to a negative cycle. Thus, we introduce *harmless* and *critical* Allen-Relations that we define as follows:

Harmless Allen-Relations cannot lead to negative cycles. In other words, a harmless Allen-Relation can be modified in any way, negative cycles can never arise from that modification. Therefore, modifications of harmless Allen-Relations cannot lead to inconsistencies in the constraint set or between the stored media data and the defined output conditions.

Critical Allen-Relations have a chance to produce negative cycles if they are modified. Thus, a modification can lead to inconsistencies, therefore, for critical Allen-Relations, a consistency check after modifications must be done.

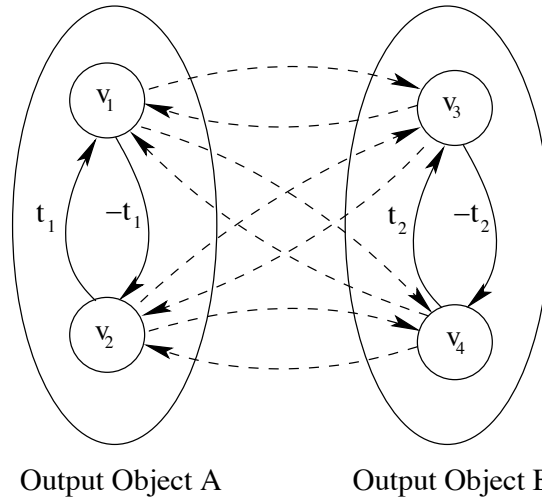


Figure 6.5: Basic Structure for Allen-Relations

Figure 6.5 shows the basic structure of Allen-Relations. The figure shows two output objects, each one has a certain interval, so each output object has a start and an end point. The interval length of the output objects is labelled as t_1 and t_2 . These edges cannot create a negative cycle on their own because they annul each other. The dashed lines depict all possible connections between the start and end points of the output objects. These edges result from difference constraints.

To generate a negative cycle, the following conditions must be fulfilled:

- At least two edges with contrary directions exist between start and/or end points of different output objects.

- At least one of these edges has a weight with $c < 0$.

Formally a necessary and sufficient condition for a negative cycle can be defined as follows:

$G = (V, E)$ is a graph with $V = \{v_1, v_2, v_3, v_4\}$ and
 $E_1 = \{(v_1, v_3), (v_1, v_4), (v_2, v_3), (v_2, v_4)\}$, $E_2 = \{(v_3, v_1), (v_3, v_2), (v_4, v_1), (v_4, v_2)\}$
 $E_3 = \{(v_1, v_2), (v_2, v_1), (v_3, v_4), (v_4, v_3)\}$, $E = E_1 \cup E_2 \cup E_3$
 The weight of all edges $e \in E_1 \cup E_2$ is $w(e) \leq 0$. The weight of edges $e \in E_3$ is:
 $w(v_1, v_2) = t_1$, $w(v_2, v_1) = -t_1$, $w(v_3, v_4) = t_2$, $w(v_4, v_3) = -t_2$.

It is important to note that the nodes v_1 and v_2 belong to an output object (A) and the nodes v_3 and v_4 belong to another output object (B). In other words, these nodes are the start and end points of the interval that belongs to an output object. The edges in E_1 are those which start in output object A and the edges in E_2 start in B . In E_3 are only those edges which connect the nodes of an output object.

The necessary condition for negative cycles is:

$$\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0 \quad (6.1)$$

The sufficient condition for a negative cycle is:

$$\exists e_1 \in E_1 : \exists e_2 \in E_2 : w_cycle(e_1, e_2) < 0 \quad (6.2)$$

The necessary condition defines that an edge from output object A to B must exist as well as an edge in the reverse direction. If an edge exists in E_1 and in E_2 , a cycle must exist that contains all nodes of the graph. This is because the edges in E_3 define two cycles, one for each output object. To make a negative cycle possible at least one of the edges between the output objects must have a negative weight.

In the sufficient condition the weight of the cycle is computed. To do this a function $w_cycle(e_1, e_2)$ is used, it takes the edges e_1 and e_2 and determines the weight of the cycle in which these edges are involved.

This formalism can be applied to restricted Allen-Relations (Table 5.1). To check the necessary condition no knowledge about quantitative properties of restricted Allen-Relations is needed. Thus, a classification of restricted Allen-Relations into harmless and critical relation can be done easily. Harmless are all those restricted Allen-Relations that do not fulfill the necessary condition for negative cycles. Critical relations are all those which fulfill the necessary condition. Table 6.1 classifies all restricted Allen-Relations as harmless or critical relation.

In the following examples we want to determine whether the restricted Allen-Relations A meets B and A overlaps(o_1, o_2) B are harmless or not. A complete examination for each restricted Allen-Relation is given in appendix A.

harmless Relations	critical Relations
$A \text{ meets } B$	$A \text{ overlaps}(c_1, c_2)$
$A \text{ starts } B$	$A \text{ overlaps}_{V_s}(c_1, l, c_2) B$
$A \text{ equal } B$	$A \text{ overlaps}_{V_e}(c_1, c_2, l) B$
$A \text{ before}(c_1) B$	$A \text{ overlaps}_{V_{se}}(c_1, l, c_2, m) B$
$A \text{ finishes } B$	$A \text{ during}(c_1, c_2) B$
	$A \text{ during}_{V_s}(c_1, l, c_2) B$
	$A \text{ during}_{V_e}(c_1, c_2, l) B$
	$A \text{ during}_{V_{se}}(c_1, l, c_2, m) B$
	$A \text{ before}_V(c_1, l) B$

Table 6.1: Classification of Harmless and Critical Allen-Relations

A meets B is the first example, it means that the interval A meets the interval B . The set of difference constraints for this Allen-Relation can be determined from table 5.1, it is: $end(A) - st(B) \leq 0$, $st(B) - end(A) \leq 0$. Furthermore, the intervals A and B have defined lengths c_A and c_B . This fact is leading to the following set of difference constraints: $\{end(A) - st(A) \leq c_A, st(A) - end(A) \leq -c_A, end(B) - st(B) \leq c_B, st(B) - end(B) \leq -c_B\}$

The next step is to build the constraint graph $G = \langle V, E \rangle$. To simplify matters the node v_0 is not part of this graph, because all edges from v_0 have the weight 0. Therefore, these edges do not need taken into account for checking the necessary condition.

Set of nodes:

$$V = \{st(A), end(A), st(B), end(B)\}$$

Set of edges:

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (end(A), st(B)), (st(B), end(A))\}$$

Edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(end(A), st(B)) &= 0, w(st(B), end(A)) = 0 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$\begin{aligned} E_1 &= \{(end(A), st(B))\} \\ E_2 &= \{(st(B), end(A))\} \\ E_3 &= \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\} \end{aligned}$$

Now the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$ must be checked. It is easy to see, that E_1 and E_2 have an element, thus a cycle must exist. The weight of both edges is 0, therefore, the necessary condition is not fulfilled. This means the Allen-Relation A *meets* B cannot produce a negative cycle. Therefore, it is a harmless relation.

A overlaps(c_1, c_2) B has the meaning that interval A overlaps interval B by at least one unit. The difference constraints assigned to this relations are: $\{st(A) - st(B) \leq -c_1, end(A) - end(B) \leq -c_2, st(B) - end(A) \leq -1\}$. Furthermore, the output objects A and B have a length which must also be defined using difference constraints. Thus, we have the following additional set of difference constraints: $\{end(A) - st(A) \leq c_A, st(A) - end(A) \leq -c_A, end(B) - st(B) \leq c_B, st(B) - end(B) \leq -c_B\}$. In the following we define a constraint graph $G = \langle V, E \rangle$ for this relation:

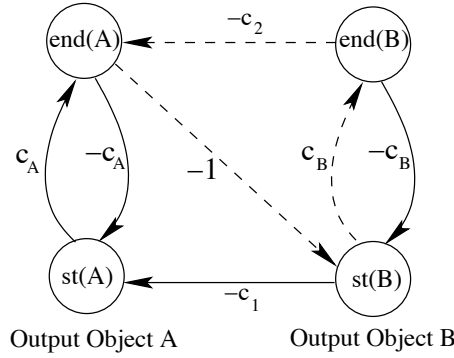


Figure 6.6: Constraint Graph for $A \text{ overlaps}(c_1, c_2) B$

Set of nodes:

$$V = \{st(A), end(A), st(B), end(B)\}$$

Set of edges:

$$E = \{ (st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), \\ (end(B), end(A)), (st(B), st(A)), (end(A), st(B)) \}$$

Edge weights:

$$w(st(A), end(A)) = c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) = c_B, w(end(B), st(B)) = -c_B, \\ w(end(B), end(A)) = -c_2, w(st(B), st(A)) = -c_1, w(end(A), st(B)) = -1$$

The edge sets E_1 , E_2 and E_3 are defined as follows:

$$E_1 = \{(end(A), st(B))\}$$

$$E_2 = \{(end(B), end(A)), (st(B), st(A))\}$$

$$E_3 = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\}$$

The necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$ have to be checked. It is easy to see, that E_1 and E_2 have an element, thus a cycle must exist. This cycle could be negative because of the fact that we have negative edges in E_1 as well as in E_2 . The dashed lines in figure 6.6 show one cycle that could potentially be negative. Thus, the relation $A \text{ overlaps}(c_1, c_2) B$ is a critical relation and it has to be checked by evaluating the sufficient condition after a modification.

6.3.3 Modifications of Output Objects

In practice we probably have more modifications of output objects than of output constraints. For checking consistency of a constraint graph only those modifications are of interest which change the lengths of the output objects. This is because these changes can produce negative cycles in the constraint graph. Figure 6.5 clarifies this circumstance. The length of the output objects is depicted by solid lines within the output objects. Not each modification on the interval of an output objects can lead to inconsistencies of the output constraints. As before, inconsistencies are detected by negative cycles in the constraint graph. Hence, the following condition must be fulfilled to produce a negative cycle:

- An edge between the start and end point of an output object must be part of a cycle.
- The weight of this cycle has a negative value.

Also for output objects we want to introduce *harmless* and *critical* output objects:

Harmless Output Objects cannot produce a negative cycle if their length changes.

Critical Output Objects can produce a negative cycle if their length changes.

If we can build such a classification, we must just check critical output objects after modifications. To introduce a formal specification for harmless and critical output objects the graph $G = (V, E)$ built from the constraint graph in figure 6.5 is used. Again the following sets of edges are used:

$$E_1 = \{(v_1, v_3), (v_1, v_4), (v_2, v_3), (v_2, v_4)\}, E_2 = \{(v_3, v_1), (v_3, v_2), (v_4, v_1), (v_4, v_2)\}$$

$$E_3 = \{(v_1, v_2), (v_2, v_1), (v_3, v_4), (v_4, v_3)\}, E = E_1 \cup E_2 \cup E_3$$

The necessary condition for negative cycles where an output object is involved is:

$$\begin{aligned} \exists e_1 \in E_1 : \exists e_2 \in E_2 : \exists (x, y) \in E_3 : w(e_1) < 0 \vee w(e_2) < 0 \vee w(e_3) < 0 \\ \wedge \text{cycle}(e_1, e_2, (x, y)) \wedge \neg \text{cycle}(e_1, e_2, (x, y), (y, x)) \end{aligned} \quad (6.3)$$

The sufficient condition for a negative cycle is:

$$\exists e_1 \in E_1 : \exists e_2 \in E_2 \exists e_3 \in E_3 : w_cycle(e_1, e_2, e_3) < 0 \quad (6.4)$$

The condition 6.3 defines that we have two edges (e_1 and e_2) with opposite directions between the output objects and one edge that comes from the length of an output object. This edge is denoted as (x, y) . All these edges must be involved in a cycle this is modelled by the predicate $cycle(e_1, e_2, (x, y))$ which becomes *true* if such a cycle exists. It is important to note that this cycle must not include both edges result from the length of an output object, this is modelled with $\neg cycle(e_1, e_2, (x, y), (y, x))$. If a cycle included both edges resulting from the output object length, the edge weight of these edges would annul each other. Therefore, the length of the output object would have no impact on the weight of the whole cycle. Furthermore, we define in condition 6.3 that at least one of the edge weights of the cycle must be negative if the whole cycle should have a negative weight.

An output object is harmless if the edges produced by the length of the output object are not involved in a potential negative cycle. In other words, modifications on an output object are harmless if condition 6.3 is not fulfilled for this output object.

Now we can start to test whether an output object is harmless or not. To do this we must check condition 6.3. We want to exemplify this on the output condition $A \text{ equal } B$. Figure 6.7 shows the constraint graph for this output condition.

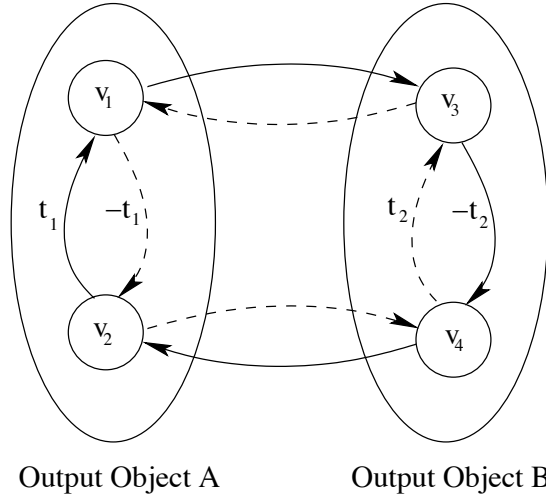


Figure 6.7: Constraint Graph for $A \text{ equal } B$

The graph $G = (V, E)$ in figure 6.7 has the following sets of edges:

$$E_1 = \{(v_1, v_3), (v_2, v_4)\}, E_2 = \{(v_3, v_1), (v_4, v_2)\}$$

$$E_3 = \{(v_1, v_2), (v_2, v_1), (v_3, v_4), (v_4, v_3)\}, E = E_1 \cup E_2 \cup E_3$$

We can find the following edges in the edge sets above: $(v_2, v_4) \in E_1$, $(v_4, v_3) \in E_1$, $(v_3, v_1) \in E_2$ and $(v_1, v_2) \in E_3$. Furthermore, we can see in figure 6.7 that there is

the cycle $\{(v_4, v_3), (v_3, v_1), (v_1, v_2), (v_2, v_4)\}$ which includes all the edges mentioned before. However, this cycle does not include the edge (v_2, v_1) . Thus, the length of output object A has impact on the weight of the whole cycle. Condition 6.3 is fulfilled and output object A is critical and must be checked after modifications. We can do the same test with the same result for output object B .

Considering the output condition $A \text{ equal } B$ again, we know from table 6.1 that the relation *equal* is harmless. Formally, this means that modifications on this relation itself cannot produce a negative cycle. In practice this relation cannot be modified because it has no parameter. We can just delete this relation if it exists. This kind of modifications cannot produce a negative cycle in the constraint graph and must not be checked. On the other side if we know that the output objects that are involved in the *equal* relation are critical. Thus, we must check the constraint graph on consistency after modifications of these output objects.

6.3.4 Algorithm for Building a Partial Graph

Before we present an optimized algorithm for building a partial constraint graph, we will summarize our findings in table 6.2. This table shows for each Allen-Relation if

Allen-Relation	critical Rel.	critical Out. Obj.	insert	update Rel.	update Out. Obj.
$A \text{ starts } B$	no	no	no	no	no
$A \text{ finishes } B$	no	no	no	no	no
$A \text{ meets } B$	no	no	no	no	no
$A \text{ equal } B$	no	yes	yes	no	yes
$A \text{ before}(c_1) B$	no	no	no	no	no
$A \text{ before}_V(c_1, l) B$	yes	no	yes	yes	no
$A \text{ overlaps}(c_1, c_2) B$	yes	yes	yes	yes	yes
$A \text{ overlaps}_{V_s}(c_1, l, c_2) B$	yes	yes	yes	yes	yes
$A \text{ overlaps}_{V_e}(c_1, c_2, l) B$	yes	yes	yes	yes	yes
$A \text{ overlaps}_{V_{se}}(c_1, l, c_2, m) B$	yes	yes	yes	yes	yes
$A \text{ during}(c_1, c_2) B$	yes	yes	yes	yes	yes
$A \text{ during}_{V_s}(c_1, l, c_2) B$	yes	yes	yes	yes	yes
$A \text{ during}_{V_e}(c_1, c_2, l) B$	yes	yes	yes	yes	yes
$A \text{ during}_{V_{se}}(c_1, l, c_2, m) B$	yes	yes	yes	yes	yes

Table 6.2: Checking for Output Constraints and Output Objects

it is a critical relation and if it includes at least one critical output object. As mentioned above checking consistency is sometimes needed after modifications on output objects or after changes on Allen-Relations. The columns *update Rel.* and *update Out. Obj.* define whether a consistency check is necessary after these modifications. Beside

modifications new Allen-Relations and output objects can be inserted in an existing constraint graph. The column *insert* indicates whether we have to check this Allen-Relation after insertion. The *delete* operation is not considered in table 6.2. Deleting an Allen-Relation cannot lead to a negative cycle in the constraint graph. In fact, deleting a relation makes the constraint set weaker. Thus, in this case no consistency check is needed. Deleting an output object also cannot lead to inconsistencies. It is assumed that all relations in which that output object is involved are also deleted. Thus, it is possible that two separate constraint graphs are the result of deleting an output object. If the original graph was consistent its two partial graphs must also be consistent because no negative cycle can arise from deleting relations.

Building a partial graph is essential for an efficient constraint checking. Therefore, we propose in figure 6.8 an basic algorithm for producing a partial graph. The findings from table 6.2 are the basics of this algorithm.

buildPartialGraph(INPUT *condition*, INPUT *operation*, IN/OUT *partialGraph*)

Input: Condition *condition*, Operation *operation*

Output: Graph *partialGraph*

```

1  if operation not DELETE {
2      if condition is Critical ALLEN Relation {
3          if condition is PartOfConstraintGraph {
4              oldCondition = getConditionFromGraph(condition.id)
5              if condition.value < oldCondition.value
6                  partialGraph.addToPartialGraph(allenrelation)
7          }
8          else partialGraph.addToPartialGraph(condition)
9      }
10 }
11 else if condition is MODIFICATION ON OUTPUT OBJECT LENGTH {
12     if outputobject is CRITICAL OUTPUT OBJECT{
13         addToPartialGraph(allenrelation)
14     }
15 }
16 return partialGraph

```

Figure 6.8: Algorithm for Building the Partial Graph

The algorithm (figure 6.8) needs the condition which is modified as input. A condition is either an Allen-Relation according to column *Allen-Relation* in table 6.2 or a length condition of an output object. With the parameter *operation* we can detect what is done with the condition (inserted, updated or deleted). We can see from line 1 in figure 6.8 that the algorithm only works for inserts or updates on conditions. In line 2 we check if a critical Allen-Relation is modified. In this case we check if we yet have

this condition in the partial graph. We include a new Allen-Relation only if it is harder as that we have yet. In line 11 we check if the condition is a modification on an output object length. If this output object is critical we must include this with the complete Allen-Relation where it is involved.

To complete the consistency check after modifications we have to execute the *Bellmann and Ford* algorithm on the partial graph. If this algorithm does not detect negative cycles in the partial graph the modifications do not have produced inconsistencies.

6.4 Consistency Check using Analytic Approach

Another approach of checking consistencies directly deals with inequalities of difference constraints. We can transpose these sets of inequalities for several terms. So, we can define upper and lower limits for lengths of output objects or we can determine limits for the parameters used in Allen-Relations.

We will give an example to clarify this circumstance. Figure 6.9 shows the Allen-Relation $A \text{ during}(c_1, c_2) B$. The set of difference constraints that belongs to this Allen-

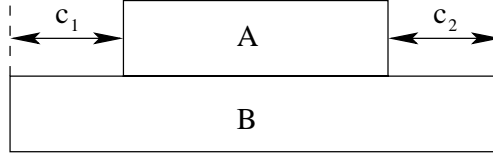


Figure 6.9: $A \text{ during}(c_1, c_2) B$

Relation is $(st(B) - st(A) \leq -c_1) \wedge (end(A) - end(B) \leq -c_2)$ (table 5.1). In this example we want to determine limits for the length of the involved output objects. Therefore, we introduce the following side conditions: $L_A = end(A) - st(A)$ and $L_B = end(B) - st(B)$. With L_A and L_B we denote the length of the output object A and B . Therefore, we have the following formulas:

$$\begin{array}{ll} st(B) - st(A) \leq -c_1 & L_A = end(A) - st(A) \\ end(A) - end(B) \leq -c_2 & L_B = end(B) - st(B) \end{array}$$

We can make the following transformation:

$$\begin{array}{ll} end(A) \leq & -c_2 + end(B) \\ -st(A) \leq & -c_1 - st(B) \\ L_A = end(A) - st(A) \leq & -c_2 + end(B) - st(B) - c_1 \\ L_A \leq & L_B - c_1 - c_2 \end{array}$$

The interpretation of the result is very intuitive when we look at figure 6.9. The maximum for the length of interval A can only be the length of B reduced by the length of c_1 and c_2 , this is because c_1 and c_2 are minimum values.

Realization within a Database System

We can analyse each Allen-Relation in the same way we have demonstrated above. Appendix B gives an overview of the results of this analysis. The advantage of this analytical approach is its easy integration into a database system because we can check consistency of output constraints by using existing features like traditional integrity constraints or simple triggers. Furthermore, these triggers and integrity constraints can be produced automatically according to our formal analysis. Because we use traditional database features the consistency check is likely to be very fast.

6.5 Auxiliary Data Structure for Checking Consistency

We use the table *Constraint_Graphs* for storing difference constraints (section 5.3). The entries of this table can be used for building a constraint graph. However, if we want to build a partial constraint graph, we need more information about the Allen-Relations used in the constraint graph. Especially, we must know whether an Allen-Relation is harmless or not. Thus, we introduce the table *Constraint_Orders* as follows:

Constraint_Orders(C_ID, predecessor, successor, R_ID)

The table *Constraint_Orders* stores information which is necessary for an optimized checking of output constraints. Especially, the attribute *R_ID* is important because it stores an identification for each Allen-Relation. This information is used to detect whether a relation between output object is harmless or not.

The attributes *predecessor* and *successor* are used for storing the structure of an Allen-Relation. Each Allen-Relation has the structure *A relation B*. We say that *A* is the *predecessor output object* and *B* is the *successor output object* of this relation. The attributes *predecessor* and *successor* store references to output objects. This information is needed for adapting output schedules.

As an example we want to show the table entries of the restricted Allen-Relation $o_1 \text{ before}_v(2, 0) o_2$ used in the example in section 5.3. The table entries which are inserted in the *Constraint_Graph* table for this output constraint are shown in figure 6.3.

<u>C_ID</u>	predecessor	successor	R_ID
c1	o_1	o_2	6

Table 6.3: Table Entries in Relation *Constraint_Orders*

Each kind of Allen-Relation from table 4.2 has a specific identifier stored in *R_ID*. So, the checking algorithm can detect whether an Allen-Relation is harmless or not. In our implementation the relation $A \text{ before}(c_1, l) B$ has 6 as identifier.

6.6 Conclusion

This section has introduced concepts for checking output consistency. A basic algorithm for checking output constraints includes the following steps:

- Building a complete constraint graph based on difference constraints. For generating the constraint graph we can use the data structure proposed in section 5.3.
- Building a partial constraint graph in case of modifications of Allen-Relations or of output objects. Auxiliary data structures can be used for building the partial graph efficiently (sections 6.5, 5.3).
- Checking consistency of the partial graph by using either a graph theoretical approach (section 6.2) or an analytical approach (section 6.4).

Both approaches for checking consistency have their rights depending on the requirements we have on the consistency check. We can summarize both approaches as follows:

The graph theoretical approach builds a constraint graph out of the a set of difference constraints. To check consistency we use the *Bellmann and Ford* algorithm to determine the shortest paths from a start node to each other node in the graph. The set of difference constraints is consistent if no negative cycles on these shortest paths can be found. While computing shortest paths a solution of the set of difference constraints is made by determining exact values for each start and end point of the intervals involved in difference constraints. A disadvantage of this approach is that it cannot use existing database features.

The analytic approach tries to find limits for the length of output objects or for the parameters used in Allen-Relations. The advantage is that we can use database features like integrity constraints or triggers to realize this approach in a database system.

Chapter 7

Management of Output Schedules

Building an output order based on defined output constraints is essential for semantical correct data output (figure 7.1). The contribution of this chapter is proposing concepts for building and adapting output schedules. Adapting output schedules is necessary after modifications of Allen-Relations or of output objects which are involved in an existing output schedule.

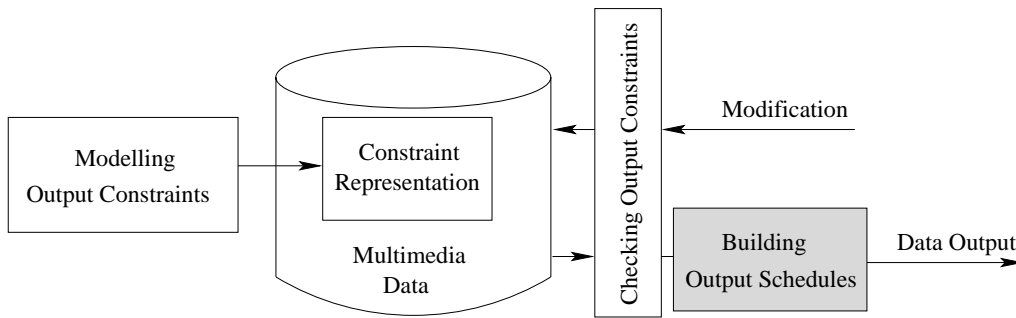


Figure 7.1: Processing of Output Constraints

According to our classification of output constraints we can build two kinds of schedules. Considering output constraints on output parameters the scheduling of resources like bandwidth is feasible. Especially, resource scheduling that considers quality of service has been investigated [BGÖS97, GÖS98, GIÖ98b]. Therefore, we concentrate on building temporal and spatial schedules according to the defined output constraints. An *output schedule* is a temporal or spatial order of output objects in such a manner that the output constraints for synchronizations defined for these output objects are fulfilled.

7.1 Use Cases for Output Schedules

The need for building output schedules arise from two different uses. We must produce output schedules based on output constraints that are defined in the database as well

as output schedules based on constraints defined in a database query. Both cases are considered in the following:

Output schedules based on output constraints represent the output order which is defined by the data producer. In practice, this schedule is used for data output if a database query does not define any output constraints. Considering our movie-example of section 4.6, we can define the query *'select * from movie where scene_nr=1'*. In this case all sound sources of the first scene are output. An output schedule is built on the basis of the output constraints defined by the data producer.

Output schedules based on database queries represent the output order which is desired by the database user. Multimedia query languages like MOQL [LÖSO97] support the specification of spatial and temporal constraints within a query. An output schedule must be built based on these constraints.

If the data producer has not defined any output constraint, the user can define arbitrary output constraints. However, conflicts can occur if the user of media data specifies output constraints which are not compatible with the output constraints defined by the data producer. There are several possibilities for handling these conflicts. A basic approach ignores one of the defined constraint sets. Thus, an output schedule is produced either based on the constraint defined by the data producer or based on the constraints defined by the data user. With a view to semantical correctness the constraints defined by the data producer should be used. However, in case of those conflicts the database user must be informed by the database system.

7.2 Producing an Output Schedule

In section 6 we have introduced a graph theoretical approach for checking consistency of difference constraint sets. This algorithm works in such a way that it takes the node v_0 as start node and computes shortest paths to all the other nodes in the constraint graph. While computing the shortest path between v_0 and another node, a weight of this path is computed.

In this chapter we use the following example: It is assumed that we have the following sound sources and lengths of sound sources in our audio scene where the sound sources are named by o_i : o_1 length(15), o_2 length(25), o_3 length(20), o_4 length(10). The relations between these sound sources are: o_1 before(2) o_2 , o_2 overlaps(3,5) o_3 , o_3 before(5) o_4 .

Figure 7.2 shows the constraint graph for this example. The computed weight for the shortest path between v_0 and a node is displayed inside that node. If we take v_0 as temporal or spatial start point, we can use the weights of the shortest paths for building an output schedule.

Figure 7.3 shows the values for the shortest paths according to figure 7.2. These values are negative. Therefore, we cannot use them directly as temporal output sched-

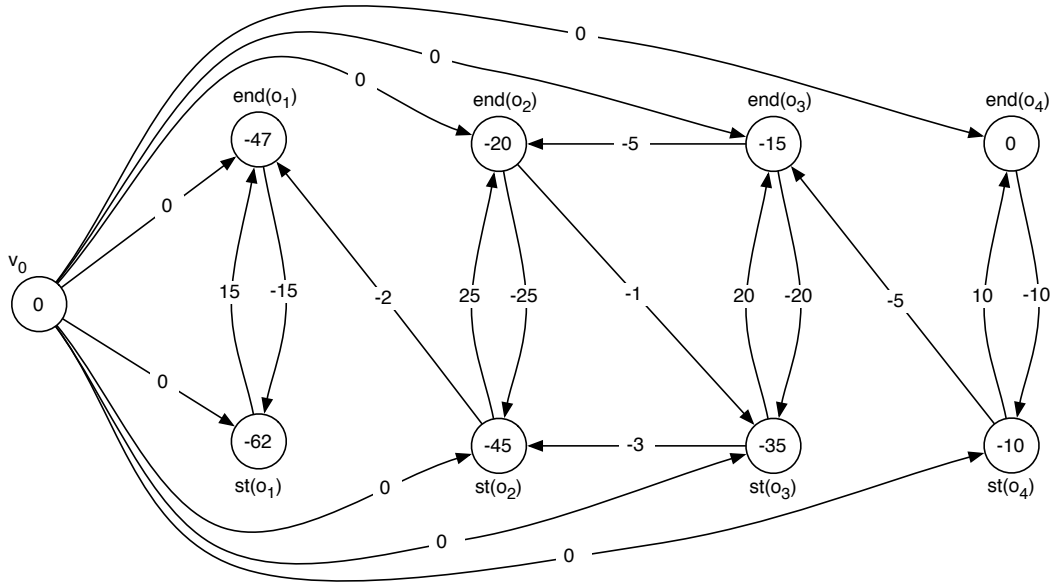


Figure 7.2: Constraint Graph with Shortest Paths for an Audio Scene

ule. Usually, we have to transform the shortest path values into a co-ordinate system or into a timeline. A temporal output schedule based on the computed shortest path

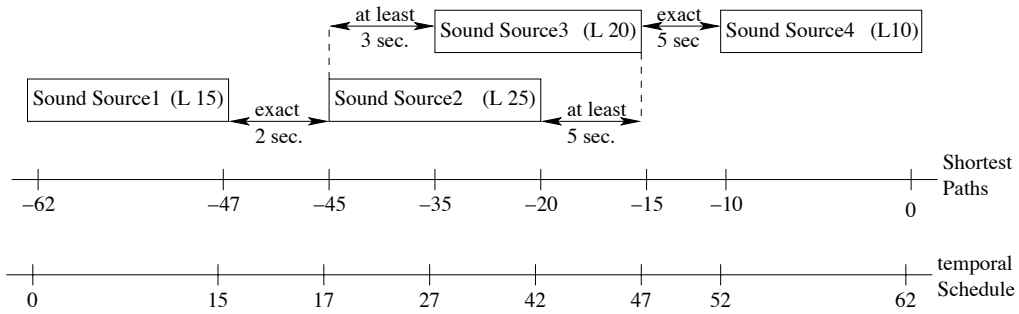


Figure 7.3: Output Schedule for Constraint Graph

weights is also shown in figure 7.3. For building this output schedule we shifted the original values of the shortest path weights into positive numbers.

To achieve this transformation we first must determine the lowest path weight (labelled as k). The interval point that belongs to that value will be the start point of the output schedule. Thus, we have to move it into the null-point of our time line. Therefore, we must apply the following transformation to each path weight L :

$$L' = L - k \quad (7.1)$$

The values computed for L' are the real start and end points of the intervals on the time line.

Usually, an output schedule is needed just before the data output process is started. Because of the fact that building an output schedule means to run the *Bellmann and Ford* algorithm on the whole constraint graph, an undesired delay can occur before data output can start. To avoid this delay output schedules can be build and adapted while checking consistency of media data after modifications. However, we have to use the graph theoretical approach for checking consistency because it computes the weights of the shortest paths additionally.

7.3 Structure of Output Schedules

For adapting output schedules we have to consider their structure in detail. Output schedules are built based on constraint graphs. These constraint graphs have some restrictions which we have already mentioned in section 6.3. For the sake of simplicity we do not consider the temporal or spatial length of output objects in the following. Thus, each output object in figure 7.4 is depicted as a node.

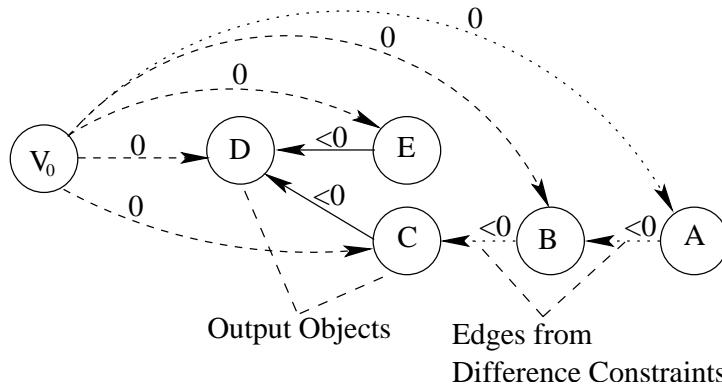


Figure 7.4: Structure of Shortest Paths in a Constraint Graph

The structure of the shortest paths between v_0 and all other nodes is mainly influenced by special properties of restricted Allen-Relations. From table 5.1 we know that all restricted Allen-Relations only have edges to their predecessor output objects (section 6.5) with weights lower than or equal to zero, while all edges to the successor output objects (section 6.5) have weights greater than or equal to zero.

Furthermore, the direct edges between v_0 and all other nodes have a weight of zero in a constraint graph. Therefore, all shortest paths between v_0 and the other nodes have the same structure. All these paths start at node v_0 and go in direction of the nodes which are the leafs of the constraint graph.

For instance, we want to consider the shortest path between v_0 and the output object C . In figure 7.4 this path is depicted with dotted lines. It is assumed that the weight of the edge between output objects A and B is negative. So, it is easy to see that the shortest path between v_0 and C is $\{v_0, A, B, C\}$. If we want to build the shortest path between v_0 and D we must consider the paths $\{v_0, A, B, C, D\}$ and $\{v_0, E, D\}$. Thus,

if we want to output all modelled output objects, the output schedule must deal with both output lines. An *output line* is a path from a leaf of an constraint graph to its root.

7.4 Consistency of Output Schedules

Consistency of an output schedule means that the order defined by this schedule is adequate for the defined output constraints. We can produce a consistent output schedule only if the stored media data are consistent to the output constraints. Hence, we have to check this before an output schedule can be produced. In this section it is assumed that an output schedule is produced once and must be adapted when media data or output constraints are modified. If we want to adapt output schedules directly after modifications, an efficient method for adapting the schedule is required.

As we have seen in section 7.3 output schedules have a definite structure. However, a modification of an output object or of an Allen-Relations have different effects on the output schedule. The position of the modified object or relation in the output line is important for its effect. Thus, we have to consider the following cases:

- Modifications of output objects or Allen-Relations which stand at the start or in the middle of an output line have impact to the rest. In other words, if we change object length or relationships between output objects on the top of the output line, the rest of the schedule must be adapted.
- Modifications of output objects or Allen-Relations which stand at the end of an output line. In this case the modification has no impact on the rest of the output schedule.

In addition to the cases mentioned above, we also have to consider output schedules with one output line as well as output schedules with many output lines. For a better understanding we will first start with a basic constraint graph that has only one output line (figure 7.2). We will explain the adaptation of this output schedule considering all possible kinds of modifications. Afterwards, we examine the adaptation of output schedules consisting of many output lines.

7.4.1 Modifying Allen-Relations in the Front Section of the Output Line

This sections considers the adaptation of an output schedule after changed Allen-Relations that stand in front or in the middle of the output line. The algorithm for adapting an output schedule in this case consists of the following steps:

- Building a partial constraint graph that only contains the modified Allen-Relation and the predecessor output object as well as the successor output object of this relation. Furthermore, the partial graph has a v_0 node which is connected to all other nodes as usual.

- Executing the *Bellmann and Ford* algorithm on the partial constraint graph and computing a correction value.
- Adapting the output schedule by this correction value.

We want to explain these steps using the example shown in figure 7.2. Because of the fact that the output objects are modelled as a list the output schedule is equal to the shortest path from v_0 to o_1 . Thus, the output schedule has the following output line: $\{v_0, o_4, o_3, o_2, o_1\}$. We modify the relationship between o_4 and o_3 because it is on the top of the output line. We want to define that the end of o_3 is exact 8 time units before the start of o_4 . Thus, we have to change the relation $o_3 \text{ before}(5) o_4$ into $o_3 \text{ before}(8) o_4$.

For adapting the output schedule, we first have to build the partial graph. The partial graph consists of the modified Allen-Relation and its predecessor and successor output object. Figure 7.5 shows the partial graph which contains the output objects o_3 and o_4 as well as the edge caused by the *before*-Relation. This partial graph is based on the constraint graph in figure 7.2. The modifications and the concerned output objects are depicted with dashed lines.

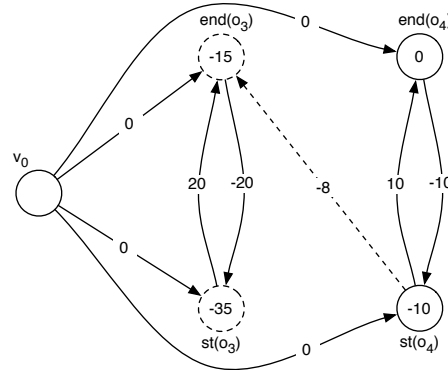


Figure 7.5: Modification in Front and in the Middle of an Output Schedule

The next step is executing the *Bellmann and Ford* algorithm on this partial graph to determine the weight of the shortest path through that graph. In practice we can get this weight of the shortest path by choosing the lowest weight of all paths. Figure 7.6 shows the partial constraint graph after computing the shortest paths. We can see that the weight of the shortest path from v_0 through the graph is -38.

The dashed boxes in figure 7.7 show the shortest paths for o_3 and o_4 before the modification. The boxes that are drawn with solid lines depict the shortest path after the modification of the *before*-Relation. In comparison with the old shortest path the new shortest path is 3 units longer. Thus, we have to shift all the other output objects which are behind o_3 in the output line (i.e. o_1, o_2) by this value.

The value that is used for adapting the output line is called *correction value*. Formally, this correction value can be determined by the difference between the weights

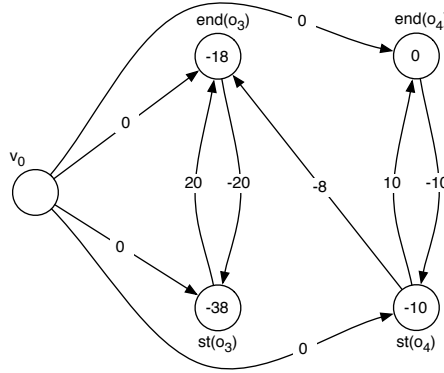


Figure 7.6: Modified Partial Constraint Graph

of the shortest path in the partial constraint graph after and before the modification. Thus, we have:

$$\text{correction value} = w(\text{shortest Path}_{\text{after B\&F}}) - w(\text{shortest Path}_{\text{before B\&F}})$$

The correction value in our example is computed as follows: $-38 - (-35) = -3$. Now we have to add this value to all weights of the shortest paths to output objects which stand in the output line behind the modification. Thus, in our example the output objects o_2 and o_1 are affected by the modification.

So, we have: $\text{end}(o_2) = -20 + (-3) = -23$, $\text{st}(o_2) = -45 + (-3) = -48$, $\text{end}(o_1) = -47 + (-3) = -50$ and $\text{st}(o_1) = -62 + (-3) = -65$. The upper part of figure 7.8 shows the whole output schedule. Compared with the original output schedule in figure 7.3 we see, that o_4 has the same position while all other output objects have been shifted by 3 unites.

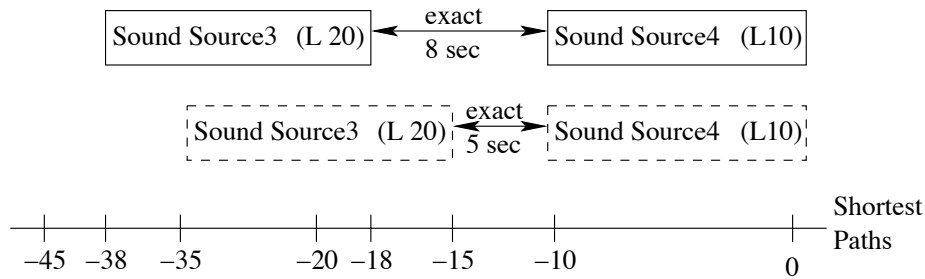


Figure 7.7: Modification on Output Schedule

As mentioned before, adapting shortest path weights affects all output objects which stand behind the modification in the output line. Another possibility of adapting an output schedule is to shift all output objects which stand before the modification in the output line. If we remember our output line is $\{v_0, o_4, o_3, o_2, o_1\}$. In our example we have modified the relation between o_4 and o_3 . Hence, we have to adapt the shortest

path to o_4 . In contrast to the approach above, we now must subtract our correction value from the weights of the considered shortest paths. For our example we have: $end(o_4) = -10 - (-3) = -7$, $st(o_4) = 0 - (-3) = 3$. The lower part of figure 7.8 shows the complete output schedule.

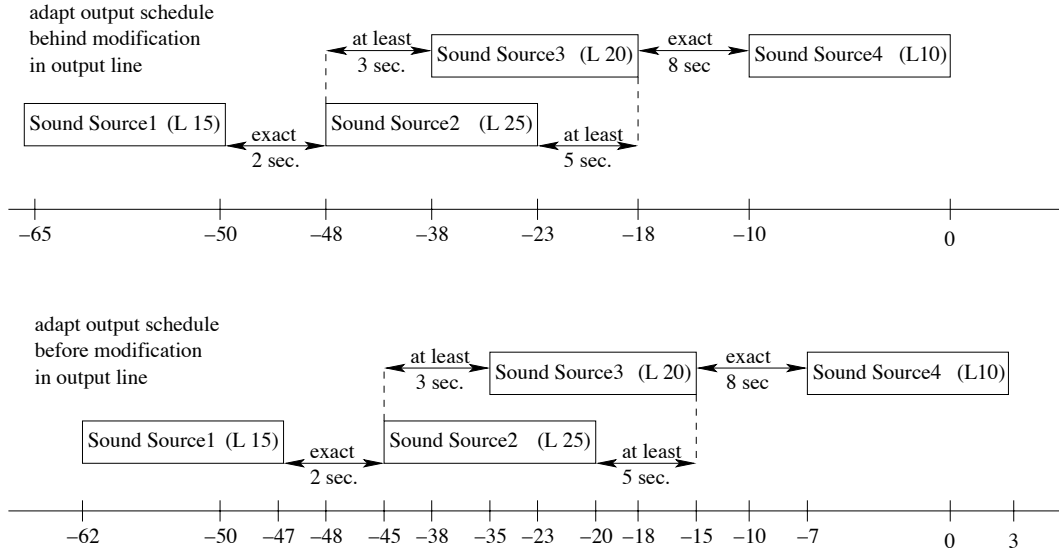


Figure 7.8: Adapted Output Schedules

Except for output object o_4 all other output object have the same position as in the original output schedule in figure 7.3. However, for $end(o_4)$ we have a positive value as weight for the shortest path. This is a contradiction to the definition of constraint graphs. Because we have an edge with weight 0 from v_0 to each other node, no shortest path can be greater as zero. To solve this contradiction we shift the whole schedule 3 units to negative and get the same result as in the upper part of figure 7.8.

7.4.2 Modifying Allen-Relations at the End of the Output Line

As mentioned before, modifications at the end of an output line have no impact on the rest of the output schedule. The output line of our example is $\{v_0, o_4, o_3, o_2, o_1\}$. Now we want to change the relation $o_1 \text{ before}(2) o_2$ into $o_1 \text{ before}(5) o_2$. Figure 7.9 shows this modification and the affected nodes with dashed lines. Furthermore, we can see in figure 7.9 the partial graph we have built in consequence of this modification.

After the *Bellmann and Ford* algorithm has been executed on this partial graph the weight of the shortest path to the nodes of output object o_1 is now $st(o_1) = -65$ and $end(o_1) = -50$. In other words, the output object o_1 has been shifted by -3 units.

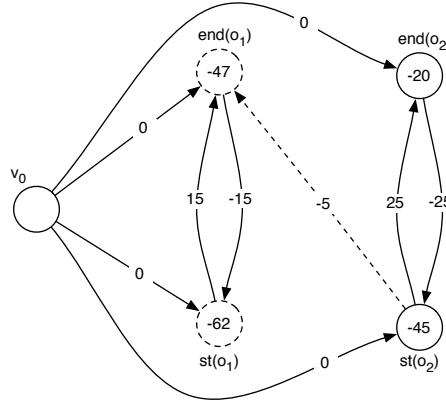


Figure 7.9: Modification on the End of Output Schedule

7.4.3 Modifications on Output Objects

Up to now we have only considered modifications on Allen-Relations. However, modifications on the length of output objects also have impact on the output schedule. We can handle this kind of modification very easily by an algorithm containing the following steps:

- Checking output consistency considering the length modification of the output object.
- Computing the value (correction value) by which the length of the output object has been modified.
- Shifting all output objects that stand in the output line behind the modified output object by using the computed correction value.

7.4.4 Output Schedules with Several Output Lines

Up to now we have focussed on output schedules for constraint graphs which have one output line. Usually, output schedules consist of many output lines. Adapting these output schedules in case of modifications of Allen-Relations or of output objects requires more attention because potentially many output lines must be adapted. We will start our explanations with an example that clarifies the problem. Afterwards, we summarize the steps required for adapting the output schedule in an informal algorithm.

Exemplification of Adapting the Output Schedule

Inserting output objects into a constraint graph can lead to constraint graphs with several output lines. We will take the constraint graph shown in figure 7.2 again as basis for our example. Assuming a new output object o_5 with length $length(o_5)$ must

be inserted. The relation between o_1 and o_5 is defined as: $o_1 \text{ before}(10) o_5$. Figure 7.10 shows the constraint graph that now contains output object o_5 as well. The produced constraint graph has the output line $\{v_0, o_4, o_3, o_2, o_1\}$ and the new output line $\{v_0, o_5, o_1\}$

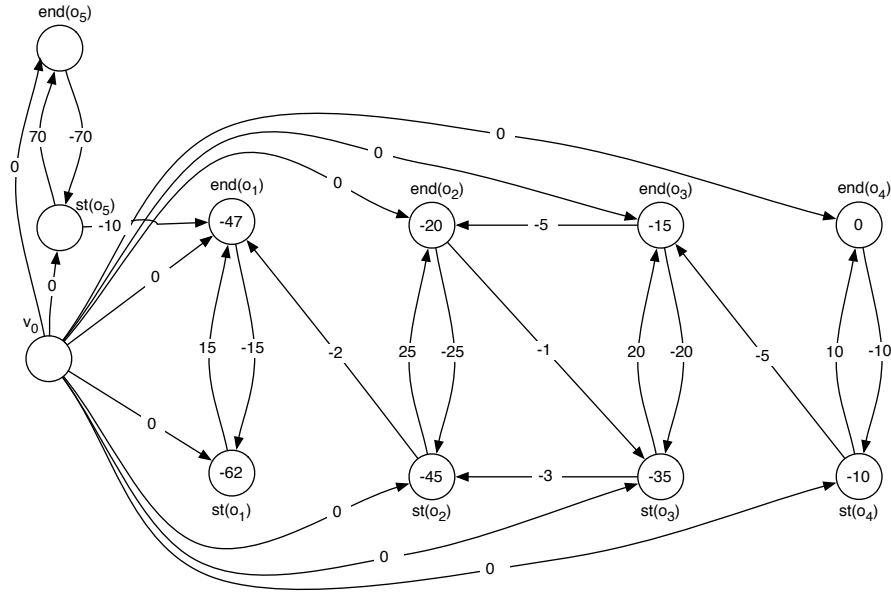


Figure 7.10: Constraint Graph with Two Output Lines

After a new output object has been integrated into the constraint graph, consistency must be checked. Considering our example we can see in table 6.2 that no consistency check is necessary, because the added *before*-Relation is harmless and cannot produce a negative cycle. However, an output schedule must be built. So, we take the new introduced relation and the output objects o_1 and o_5 to build a partial graph which is shown in figure 7.11. After we have computed the new shortest path for o_1 , nothing

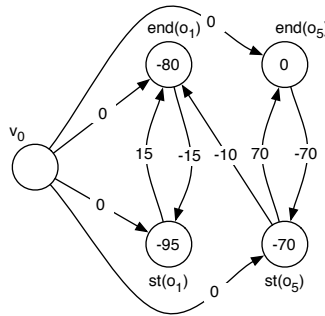


Figure 7.11: Output Schedule for new Output Line

else must be adapted in the new output line. However, output object o_1 is involved

in the output line $\{v_0, o_4, o_3, o_2, o_1\}$ as well. Thus, we must check whether the new shortest path for output object o_1 is adequate for this output line as well. The upper part of figure 7.12 shows the output schedule for the new output line, the lower part shows the already known output schedule for the original output line. It is easy to see that the new position of output object o_1 is not adequate for the original output line because the relationship $o_1 \text{ before}(2,0) o_2$ cannot be fulfilled.

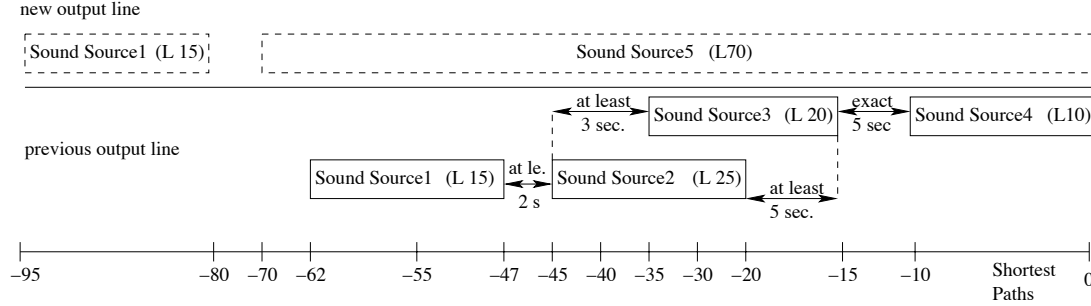


Figure 7.12: Output Schedules for Both Output Lines

Thus, we have to shift all output objects of the original output line in such a way that the output schedule becomes consistent. This generally means if the weight of the shortest path of an output object is changed, all output lines in which this object is involved must adapt their shortest path weights.

To adapt the weights of the shortest paths we again use a correction value which is built as shown in section 7.4.1. In our example the weight for the shortest path to o_1 before the modification was -62 (figure 7.10). After the modification we determine a shortest path with a weight of -95 (figure 7.11). The correction value is $-95 - (-62) = -33$. With this value we move all output objects in the original output line. So, we get the following values: $st(o_2) = -45 + (-33) = -78$, $end(o_2) = -20 + (-33) = -53$, $st(o_3) = -35 + (-33) = -68$, $end(o_3) = -15 + (-33) = -48$, $st(o_4) = -10 + (-33) = -43$, $end(o_4) = 0 + (-33) = -33$.

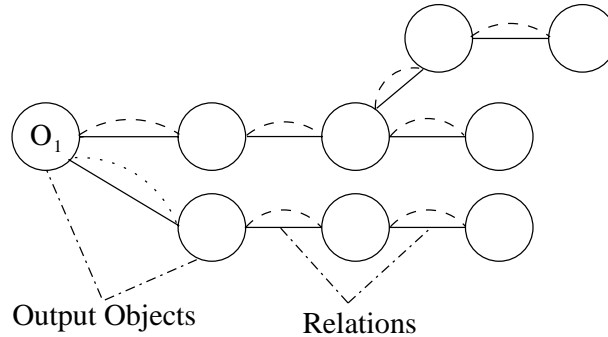


Figure 7.13: Output Schedule with Sveral Output Lines

Figure 7.13 depicts possible consequences of a modification in a constraint graph. We assume that we change the Allen-Relation depicted by the dotted line. The dashed

lines show the required adaptations that are necessary in consequence of modifying the Allen-Relation. As usual, all output objects which stand behind the modification in the output line must be adapted. Furthermore, we have to check whether modifying the weights of the output object that stands on top of two output lines (O_1) leads to inconsistencies in any of these output lines. In case of inconsistencies we have to adapt the affected output line as well.

Algorithm for Adapting Output Schedules with Several Output Lines

For adapting output schedules consisting of many output lines the following steps are required:

- Adapting the output line where the modification takes place (sections 7.4.1, 7.4.2, 7.4.3).
- If we must adapt the weights of an output object that is the predecessor output object of more than one Allen-Relation, consistency must be checked for each of these Allen-Relations.
- If inconsistencies occur, the affected output line must be adapted as well.

7.5 Checking Output Consistency During the Output Process

Managing output schedules includes techniques to guarantee output constraints during the output process. This means we have to check the output constraints on output parameter and synchronization constraints during the data output process. If these output constraints are violated, an action defined in the output constraint will be executed.

We have done several work in this field. In [Gul05] an approach is proposed to check output constraints defined on dynamic output parameters. Actually, output constraints which define the required bandwidth of a video are observed during the video output by the database. For building this video database we use ORACLE and its object-relational features. We enhanced the original video type. This enhancement can handle special types of output constraints. We stored our videos in several versions each of them optimized for a specific bandwidth. We made simulations for several bandwidth and we changed the bandwidth during the output process. The bandwidth was triggered and our output constraints react in such a way that always that video was chosen which was optimized for the available bandwidth. We could show that the video quality on the user side was much better as the normal video output. Especially when many videos were output equally each user got a much better video quality.

Another work [Rus05] deals with spatial and temporal synchronization constraints for audio data. We use this approach for wave field synthesis in cinemas. We have to guarantee that a specific slice of audio data reaches the right renderer (spatial position)

in a certain time. So, we use the temporal and spatial output schedules for generating a special data organization for audio data of movie scenes. It is shown in [Rus05] that with this special data organization all spatial and temporal output constraints for wave field synthesis can be guaranteed. An additional checking of output constraints during the output process is not necessary.

7.6 Conclusion

Beside checking output consistency during data input and when manipulations occur we must consider output consistency for the (real) data output. In practice this means an output schedule must be built which defines a specific output order. This can be a temporal or spatial order as well as an order based on resource requirements.

We introduced a method to build output schedules for temporal and spatial output constraints. Our method is based on the algorithm of *Bellmann and Ford*. The advantage of our approach is that we can use the same algorithm for checking consistency during data input as well as for producing output schedules. Furthermore, the weights of the shortest paths, which are required for producing output schedules, are computed while checking consistency.

Modifications of output constraints or of output objects have an impact on the output schedule. We have seen that the partial graph used for producing a new shortest path only contains the modified relation and its output objects. A correction value must be built based on the weights of the old and new shortest path. Weights of shortest paths must be adapted using this correction value. The position of the modification in the output line defines which other weights of the shortest paths must be adapted. If we consider output schedules with several output lines correction of weights can be run through different lines.

Chapter 8

Implementation and Evaluation

This section wants to prove the practical usage of the concepts proposed in this thesis. We consider first qualitative properties concerning modelling and handling of output constraints. Thus, a concrete implementation of output constraints for a wave field system is considered [HSRG05]. We will check how the requirements specified in chapter 2 can be satisfied with our approach.

Another part of this chapter deals with quantitative test. We have implemented the proposed algorithms for checking output constraints and adapting output schedules. We will prove with experiments that we can execute output constraint checking and schedule building so fast that it can be used in a real database system.

8.1 Implementation of Output Constraints for Wave Field Synthesis

Figure 8.1 shows the process of sound production for a wave field system. With this process we want to produce spatial sounds for films. The depicted process comes from today's practice where film scenes and audio scenes are not linked. During the film presentation several devices are used to play sound and movie synchronously.

The first step is the *modelling* of audio scenes. An audio scene is composed from audio objects. Each audio object is parameterized by properties like name, loudness, time of playing and further metadata. Output constraint on output parameters, like loudness, can be used for restrictions during the data output. Spatial and temporal synchronizations are another important point for modelling audio scenes.

Furthermore, modelling includes recording and storing audio objects. We store meta data, output constraints and audio data together in a database system. Only audio objects which are appropriate to our output constraints should be stored. This means we have to check consistency between output constraints and audio data during the data input or modifications.

During the output process sound sources are produced on specific places in the listening room. Renderers are special components which control the loudspeakers in

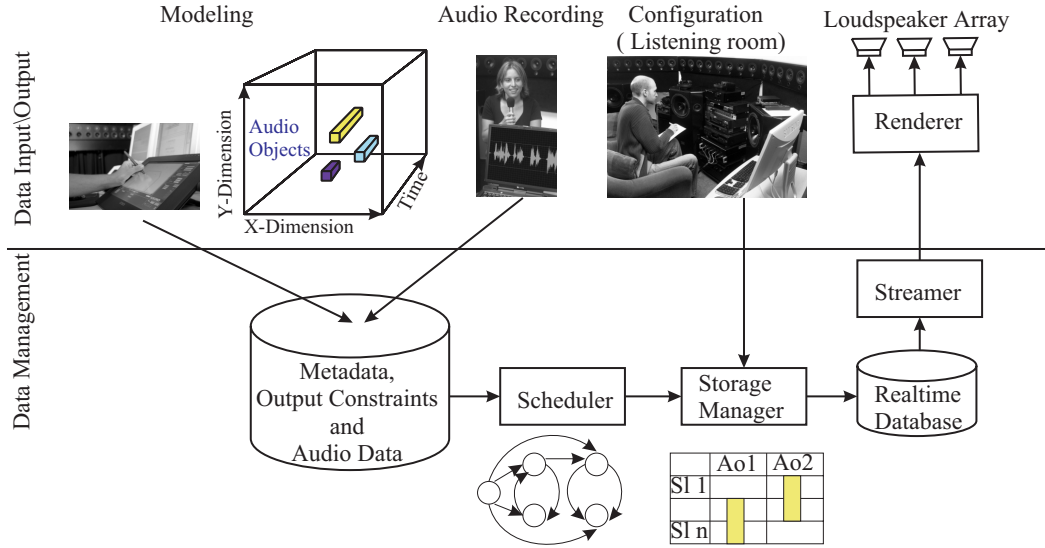


Figure 8.1: Process of Sound Production

the cinema. Actually, the renderers build the wave filed. They need a certain amount of audio data within a defined period. Hence, the spatial and temporal output schedules are used to build a specific data organization. With it we can guarantee a data output according to the defined output constraints.

8.1.1 Modelling of Output Constraints

During the modelling phase we want to define spatial and temporal output constraints for sound sources. Beside the output constraints we have to store the required audio data into the database. We use the following table for storing these audio objects:

CLIP(SCENE_NR, SOUND_SOURCE_1, SOUND_SOURCE_2, SOUND_SOURCE_3)

We used a similar relation schema for the examples in chapter 4. It is assumed that the following row exists in the table *CLIP*:

SCENE_NR	SOUND_SOURCE_1	SOUND_SOURCE_2	SOUND_SOURCE_3
1	violin	trumpet	saxophone

Table 8.1: Audio Objects for a Video Scene

We have a scene with $SCENE_NR = 1$ and one audio object for each sound source. All audio objects have a complex data type. Thus they are contains metadata, like the audio length, as well.

For this basic example it is assumed that the stored audio data can be output without any format transformation. However, we have to build output objects from the stored audio data that have spatial dimensions which are generated by an output function.

Figure 8.2 shows the spatial output constraint we want to define. This output constraint should be defined only for the first audio scene. Therefore, we use in figure 8.2 the concrete audio objects *violin* and *trumpet* instead of the attribute names. The spatial output constraint deals not with the audio object *saxophone*. Thus, it has no defined spatial position in that audio scene. Its sound is constantly played by each loudspeaker. Sound sources have no spatial dimension because they are point objects.

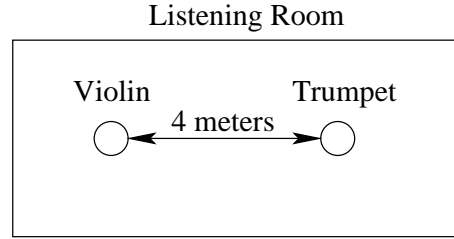


Figure 8.2: Spatial Output Constraint

Thus, if we define a spatial output constraint, no LENGTH-clause is necessary. The following definition can be made for the output constraint shown in figure 8.2:

```

CREATE OUTPUT_CONSTRAINT ViolinWestTrumpet
  CHECK          not exists (select * from clip
                                where scene_nr=1 and not
                                (play(sound_source_1) west(4,0) play(sound_source_2)))
  REASON         output
  REACTION OUTPUT          notification

```

The definition above uses the DDL notation for output constraints introduced in section 4.7. The output condition uses the output function *play()* that takes a sound source and plays it without any transformations. It is defined that the audio object of *trumpet* must be exactly 4 units west of the audio object of *violin*. The output condition must be checked only for data output and in case of a violation the user gets a notification as reaction.

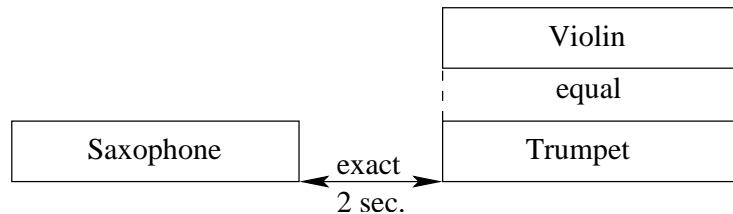


Figure 8.3: Temporal Output Constraint

Figure 8.3 shows the temporal output constraint for the audio objects in the first scene. We define no specific length for the audio objects. The length must directly be determined by using the audio data stored in the database.

Beside defining output constraints using a DDL statement, we can use a special XML structure as well (section 4.7). Thus, we define the temporal output constraints shown in figure 8.3 by using the following XML structure:

```

< OutputObject >
  < Name > o1 < /Name >
  < Value > select sound_source_3 from movie where scene_nr=1 < /Value >
< /OutputObject >
< OutputObject >
  < Name > o2 < /Name >
  < Value > select sound_source_2 from movie where scene_nr=1 < /Value >
< /OutputObject >
< OutputObject >
  < Name > o3 < /Name >
  < Value > select sound_source_1 from movie where scene_nr=1 < /Value >
< /OutputObject >
< OutputFunction > play < /OutputFunction >
< Conditions >
  < Constraint name = "length" >
    < Objekt > o1 < /Objekt >
    < Parameter > select sound_source_3.getLength() from clip where scene_nr=1
    < /Parameter >
  < /Constraint >
  < Constraint name = "length" >
    < Objekt > o2 < /Objekt >
    < Parameter > select sound_source_2.getLength() from clip where scene_nr=1
    < /Parameter >
  < /Constraint >
  < Constraint name = "length" >
    < Objekt > o3 < /Objekt >
    < Parameter > select sound_source_1.getLength() from clip where scene_nr=1
    < /Parameter >
  < /Constraint >
  < Constraint name = "beforev" >
    < Objekt > o1 < /Objekt >
    < Objekt > o2 < /Objekt >
    < Parameter > 2 < /Parameter >
    < Parameter > 0 < /Parameter >
  < /Constraint >
  < Constraint name = "equals" >
    < Objekt > o2 < /Objekt >
    < Objekt > o3 < /Objekt >
  < /Constraint >
< /Conditions > < Reason > modification < /Reason >
< ReactionModification > notification < /ReactionModification >

```

In the structure above we use select statements to detect the required audio data and their temporal length. As in the DDL expression we use the output function *play()* which simply plays the stored audio data without any modification. The real output condition is realized by the restricted Allen-Relations *before_v* and *equals*. We check this output constraint every time we modify one of the audio objects involved in the output condition. The reaction in case of a constraint violation is to notify the user.

As a result of the modelling phase the data producer has defined output constraints in a descriptive way. For our prototype implementation we use the XML structure for defining output constraints. We made this decision from practical reasons, since a syntactical and semantical checks can be implemented easily for an XML structure.

8.1.2 Integration of Output Constraints into a Database System

A special component called *transformer* takes the XML file from the modelling tool and integrates the output constraints into the database system. Actually, the transformer parses the XML file and builds difference constraints for all defined relations between output objects. Furthermore, the transformer fills these difference constraints into special tables which are used for managing output constraints. Thus, these tables are a form of ‘system tables’ which we require for our implementation of output constraints. We have introduced these tables in section 5.3 and section 6.5 as follows:

Constraint_Graphs(ID, start_node, end_node, weight, C_ID)
Constraint_Orders(C_ID, predecessor, successor, R_ID)

However, the transformer must also support building an XML file containing descriptive definitions of output constraints based on the table entries of the tables mentioned before. The modelling tool must be able to import the XML file for further editing of output constraints.

8.1.3 Checking Output Constraints

The wave field synthesis requires checking of output constraints after modifications on audio data as well as after changes on output constraints. For checking output constraints after modifications the length of the audio data is triggered. Thus, *before insert triggers* are produced for all audio data which are involved in output constraints. These triggers fire whenever the duration of audio data is changed. As a result the *length* constraints in table *Constraint_Graphs* are changed. Length modifications of audio data lead to changes of weights in the corresponding difference constraints. Thus, the corresponding attribute *weight* in table *Constraints_Graph* is changed. Furthermore, a consistency check of the affected output constraints is started (chapter 6). If a data modification violates an output constraint the reaction defined in the definition of the output constraint is executed.

8.1.4 Producing Data Output

For generating a specific wave field for a cinema, output schedules for each dimension must be built. In our example a temporal output schedule and two spatial output schedules are required. A specific characteristic for wave field synthesis is that the spatial measures of the listening room are important for the spatial output schedule. For building the final spatial output schedule we must first generate an output schedule as usual. This schedule must be modified considering the specific room properties. Therefore, a generic sound production is not possible. The wave field must be rendered for each cinema configuration to get an optimized wave field for the cinema.

For sound production we have to support audio output during the modelling phase as well as the final data output of the whole movie sound. During the modelling phase the sound designer must test several sound impressions. Therefore, database queries are sent from renderers to the database system to get several parts of audio data. The following query gives an example of such a query:

```
SELECT * FROM clip,position WHERE clip.scene_nr='1' and
position.renderer_position=P1
```

The table *position* has the following structure:

```
POSITION(SCENE_NR, SOUND_SOURCE, RENDERER_POSITION)
```

This table stores the spatial position of sound sources which are generated by the spatial output schedule. The result of the query above are all sound sources from scene '1' which belong to the renderer on position *P1*. The sound sources must be output according to the temporal output schedule.

During the output of movie sound in a cinema processing database queries is impossible because of performance requirements. In practice a renderer needs 512 audio samples in 10 ms for each sound source. Each audio sample is coded with 24 Bit, thus, 12 KBit must be output for each sound source. Because an audio scene has many sound sources a very high output rate is required. Therefore, we generate a special output file on the basis of the output schedules.

By using the output schedules a scene graph can be built which shows the logical structure of the audio scene. Figure 8.4 shows a scene graph for our example. In addition to the time and space values determined from temporal and spatial output constraints, values of output parameters like loudness are part of the scene graph as well. These values can be restricted by output constraints on output parameters.

Database tables are used to store audio data corresponding to the scene graph. Figure 8.5 shows the logical structure of a table for the temporal output schedule in our example. Each sound source is divided into slots of 512 audio samples which is equal to 10 ms audio output. Such sparse tables based on the spatial output schedules are generated as well.

The rendering process of the movie sound for a specific cinema generates a physical data output file (Figure 8.6) based on the logical structure stored in the database.

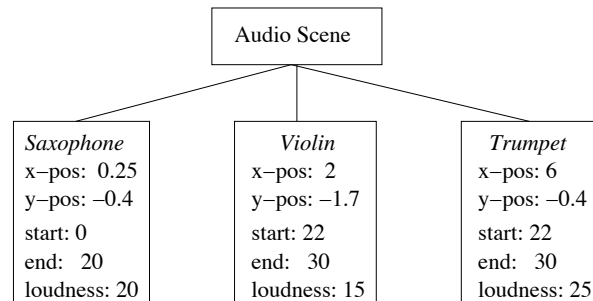


Figure 8.4: Scene Graph for Audio Scene

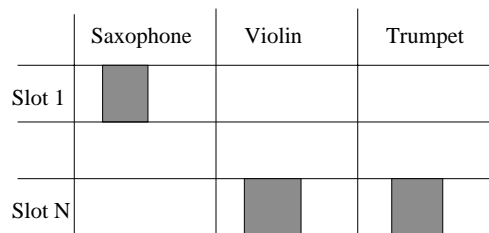


Figure 8.5: Sparse Table for Sound Sources

The data are stored in playtime order, thus at runtime the file can usually be chronologically read without jumps in any direction.

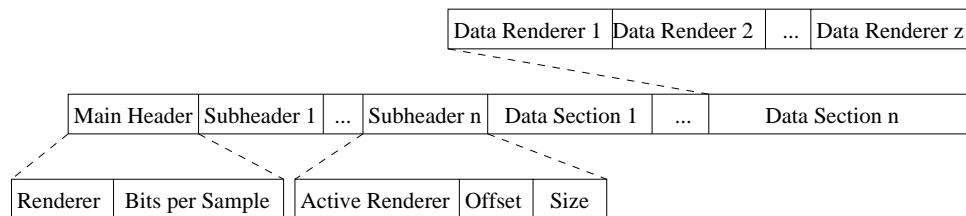


Figure 8.6: Physical Data Output File

The data output file has exactly one main header defines file-wide parameters like bits per second or the maximum channel number. The several sub sections per file define the data output for an output slot (fig. 8.5). A subheader provides section-specific information that is a list of renderer the section stores data for as well as its offset and size. The data section itself contains the slots of audio data (each slot is 512 audio samples), ordered by renderer number and playback time. Due to this structure, the data can sequentially be read during normal playback. The several audio slots must just output to the corresponding renderers.

8.2 Implementation of Further Scenarios

In addition to the wave field scenario we have introduced in chapter 2 application scenarios dealing with virtual universities and hospital database applications. We only considered the implementation of the wave field scenario in detail because handling of output constraints is similar in each application scenario. In contrast to wave field synthesis the other application scenarios deal not only with audio data. However, modelling, database internal representation, and checking of output constraints can be implemented similarly. There is a difference between these application scenarios considering data output. For wave field synthesis we have generated a special data output file which is based on the output schedule. However, for the other application scenarios we have to produce a data stream for data output.

8.3 Experiments

Main points in this work deal with efficient methods for checking output constraints and producing output schedules. Therefore, we want to prove the efficiency of the developed methods with experiments. All experiments have the same structure. We use constraint graphs of different sizes and make modifications on these graphs. We want to test how efficiently output consistency can be checked or how long it takes to adapt the output schedule. We want to show that output constraints can be implemented into a multimedia database system without any noticeable delay for query processing or data output.

The constraint graphs used for the experiments have 50, 100, 250, 500, and 1000 source objects. These are media objects (e.g. sound sources) which can be seen as temporal intervals with start and end points. These points are represented by nodes in the constraint graph. Additionally the node v_0 is needed in each graph. Therefore, the graphs have 101, 201, 501, 1001, and 2001 nodes. The relationships between the nodes are restricted Allen-Relations which are produced randomly. However, we used the same constraint graphs for all experiments. For the experiments we modified each constraint graph 1, 25, 50, and 100 times randomly.

The algorithm for checking output consistency and building output schedules are implemented in Java and were run on an iBook G4 with 768 MB main memory. We only considered the case that the constraint graph is already in main memory. Thus, no access on hard-disk is necessary during constraint checking. We can make this prerequisite because even the largest constraint graph, with 1000 source objects, only needs 100 MB of main memory. In practice the aforementioned tables *Constraint_Graphs* and *Constraint_Order* must be read for building a constraint graph. However, this time depends on the system used and has no impact on the real constraint checking algorithm.

The range of our experimental results was very large. Thus, we must choose a logarithmic scale for depicting the results. Some quantities could not be measured

directly because there are too small. Therefore, we took the time for 10.000 runs of the experiments and computed the result for a single run.

Evaluation of Output Constraint Checking using the Analytical Approach

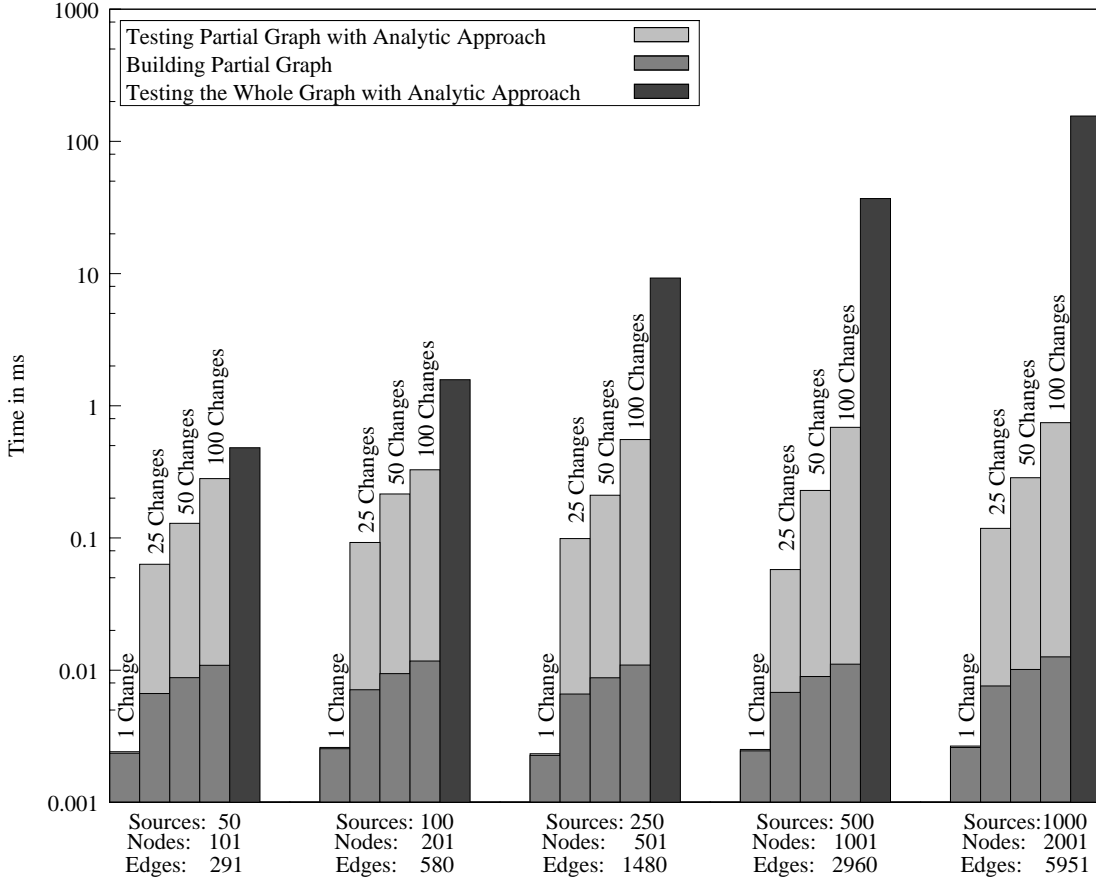


Figure 8.7: Checking Output Consistency with Analytic Approach

The first experiment is to investigate the efficiency of checking output constraints using the analytic approach shown in section 6.4. This approach can check output consistency by checking sets of inequalities (Appendix B) which deal with the length of an output object. We want to show in this experiment that building a partial graph and running the analytic consistency check on it is more efficient than running the analytic consistency check on the whole constraint graph.

To realize this experiment we modified our constraint graphs and ran a complete constraint check on each constraint graph and determined the required time. Furthermore, we determined the time for generating partial graphs on basis of the original constraint graphs after their modifications. Thereafter, we executed the analytic constraint check on these partial graphs and took the time.

The analytic constraint check can be executed very fast. In contrast to this building a partial graph is much more complex. However, we expected that building partial graphs and running the analytic constraint check on it is faster than checking the whole constraint graph.

Figure 8.7 shows the results of this experiment. We see that building a partial graph and checking it for consistency is in any case faster than checking the whole graph. In case of a single change on the constraint graph we cannot see the required time for constraint checking in figure 8.7. This is due to the fact that for a single change just a few inequalities must be checked which is so fast that it cannot be depicted in our diagram.

Evaluation of Constraint Checking by using the Bellmann and Ford Algorithm

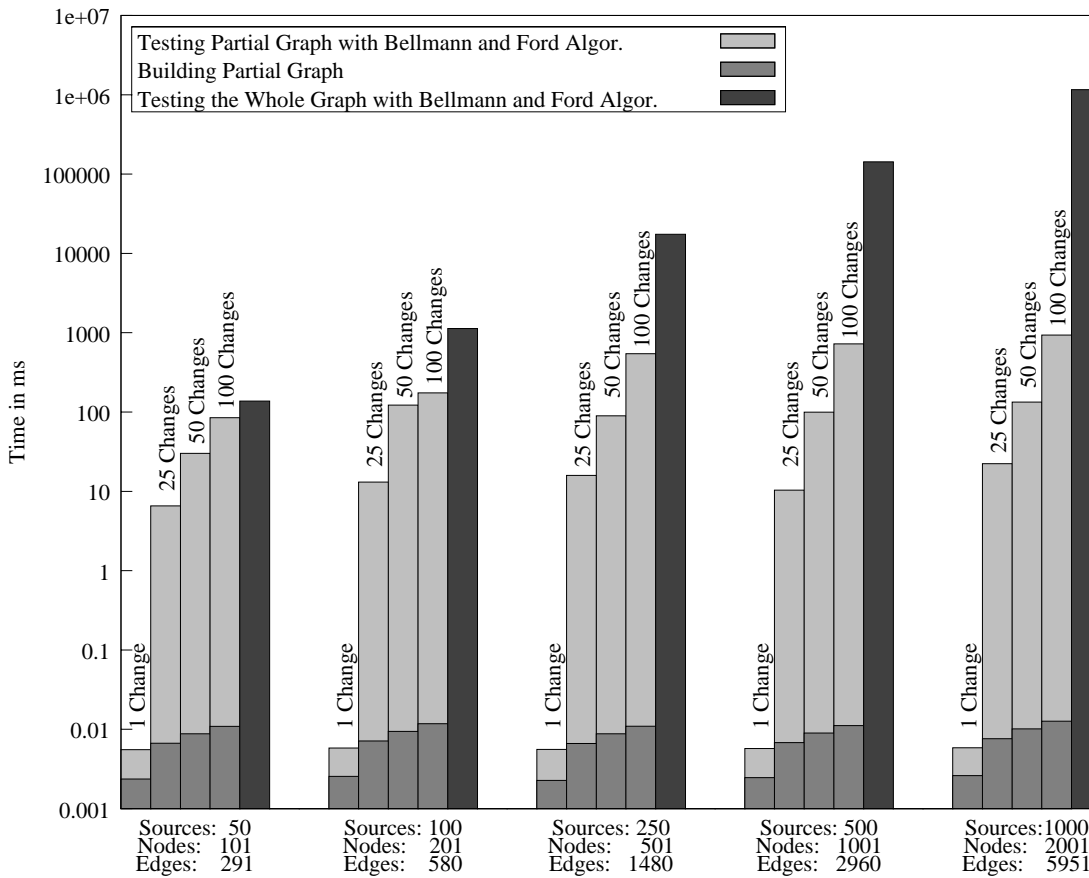


Figure 8.8: Constraint Checking with the Bellmann and Ford Algorithm

Another way of checking output consistency is using the Bellmann and Ford Algorithm (section 6.2). This approach checks consistency of a constraint graph by determining the shortest paths between v_0 and each other node in the graph. For this

approach we also want to show that building a partial graph and checking it is faster than checking the whole graph.

For realizing this experiment we modified our original constraint graphs as mentioned above. We took the time for running the Bellmann and Ford Algorithm on the whole graph as well as executing it on the partial graph.

In contrast to building a partial graph, the Bellmann and Ford Algorithm is more complex. Therefore, we expected that running the Bellmann and Ford Algorithm on the whole graph is slower than building a partial graph and run the algorithm on it.

The results of this experiment have shown that building and checking the partial graph is much more efficient as checking the whole graph (figure 8.8). In comparison to the analytic approach for constraint checking the required time for the Bellmann and Ford Algorithm exceeds the time of the analytic approach by several orders of magnitude. However, the Bellmann and Ford Algorithm also brings acceptable results because we can even check high numbers of changes in very large constraint graphs in less than a second.

Evaluation of Adapting Output Schedules

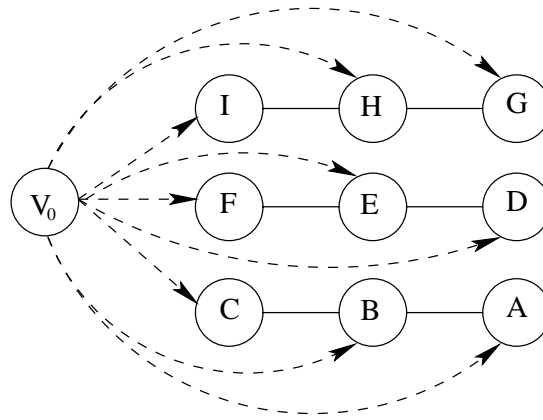


Figure 8.9: Parallel Structure of Constraint Graph

In addition to checking output consistency we have to adapt the output schedule. In practice changes of the length of output objects or of Allen-Relations can make the constraint graph inconsistent. Furthermore, the absolute output order can be changed as well. A possibility for checking consistency as well as for producing an output schedule is to run the Bellmann and Ford Algorithm on the whole constraint graph. However, section 7.4 proposed another method which generates a partial graph, checks its consistency, and adapts the output schedule using correction values.

The goal of this experiment is to determine whether our proposed method is more efficient than generating a new output schedule after modifications on the constraint graph.

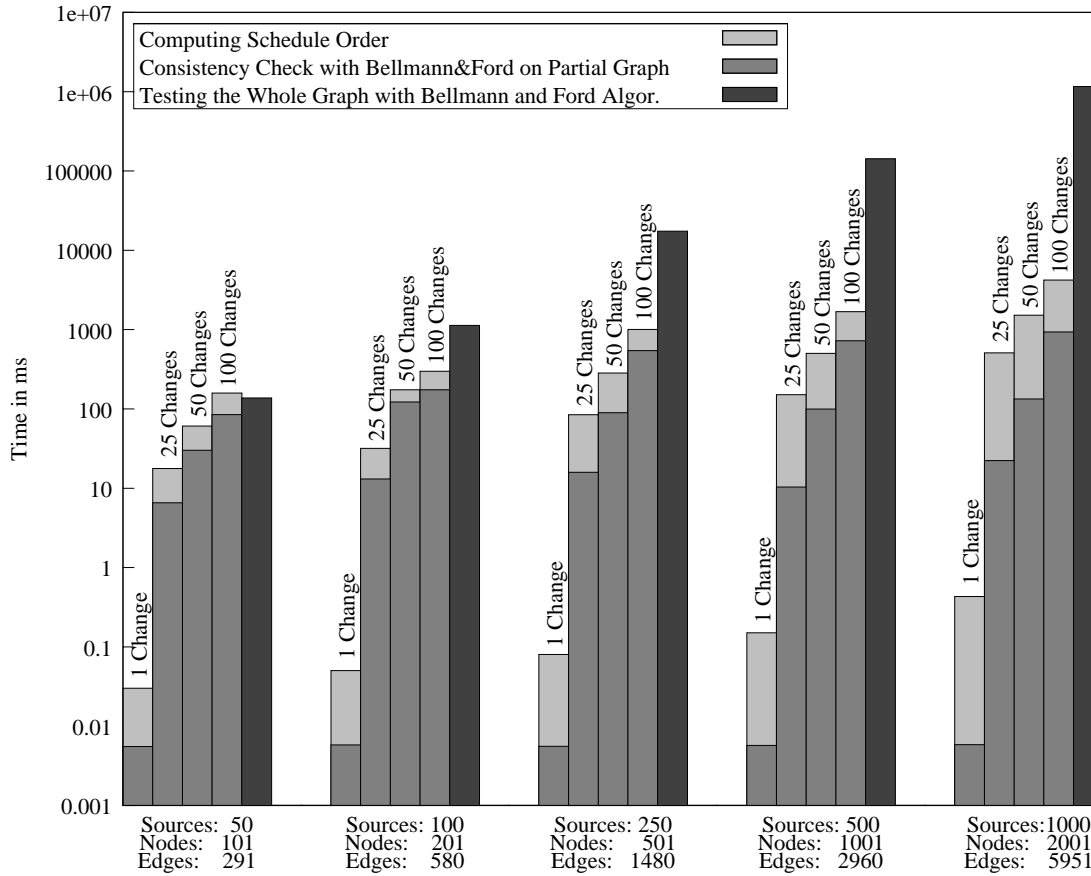


Figure 8.10: Adapting Output Schedules with Independent Output Lines

To realize this experiment we modified the constraint graphs as usual. For building a completely new output schedule we run the Bellmann and Ford Algorithm on the whole graph.

For adapting an output schedule using a correction value we need the original schedule graph before the modifications were made. It is assumed that this schedule is available. So, we can build an adapted output schedule by generating a partial graph, check its consistency, and computing the correction value.

However, we have to consider the structure of the constraint graphs because it has impact on the number of corrections we have to do for building a new output schedule. For this experiments we use a graph structure with several parallel output lines. Figure 8.9 shows the output objects *A* to *I* which are arranged in three parallel output lines. This graph structure has the advantage that the adaptation of an output schedule has no impact on the output schedule of another output line. Thus, the correction of the output schedule values is restricted on the output line where the modification takes place.

The adaptation of output schedules takes a long time, because it depends on the position in the constraint graph where the modification is made. However, we expected that adapting output schedules using correction values are more efficient than building

a completely new output schedule for the whole constraint graph.

Figure 8.10 shows the results of this experiment. Mostly, computing a new output schedule by using correction values is much faster than producing a completely new output schedule by using the Bellmann and Ford Algorithm. The advantage of using correction values becomes very big if the constraint graph becomes larger. However, there is a situation where computing a complete new output schedule brings better results. We can see this on the values for 100 changes on a constraint graph with 101 nodes. If the number of changes is so high that nearly the whole graph is changed, the partial graph is almost equal to the original constraint graph. Therefore, checking output consistency takes a long time. Furthermore, we have to adapt the original output schedule. All these facts lead to a loss in efficiency.

Adapting Output Schedules Considering their Structures

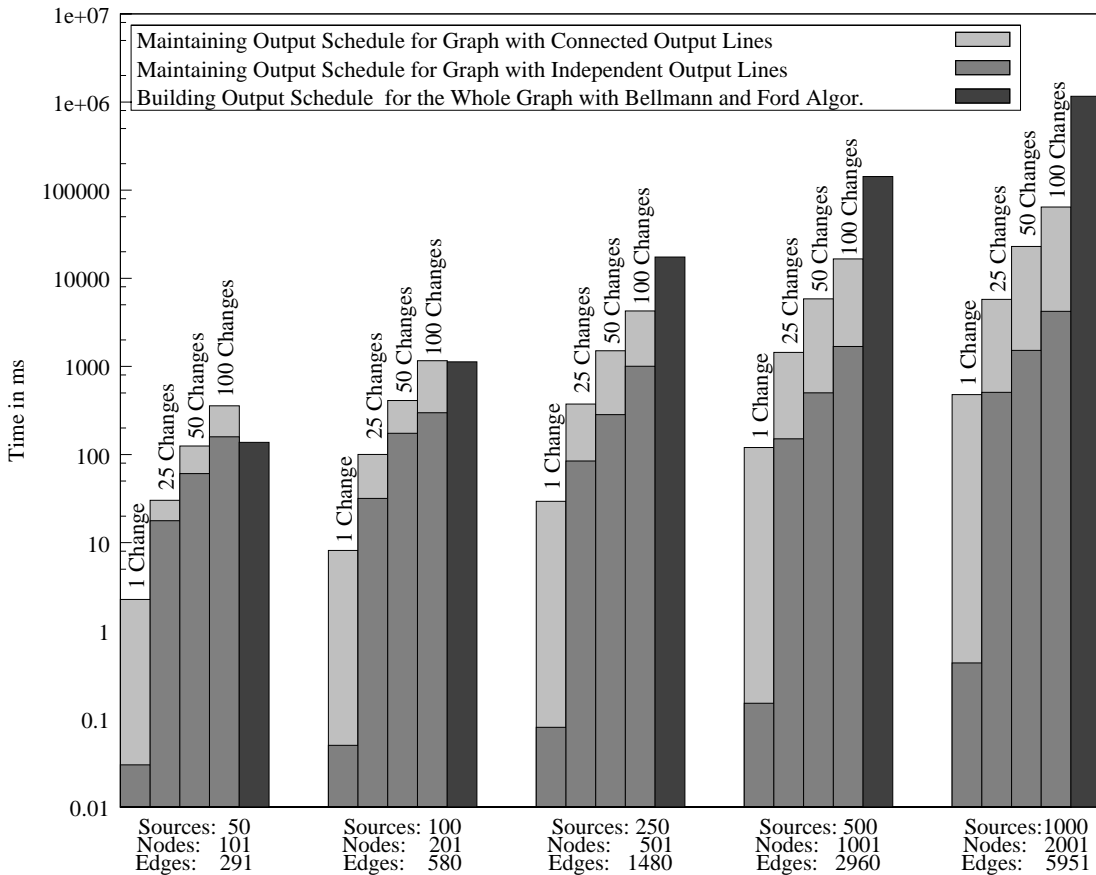


Figure 8.11: Adapting Output Schedules Depending on Constraint Graph Structure

In the above experiment we used a constraint graph structure which is very suitable for adapting output schedules by using correction values. Therefore, we want to make a test with a graph structure that represents the worst case for adapting output schedules.

Figure 8.12 shows this structure. In contrast to figure 8.9 we have a central output object named X which connects all output lines. Thus, we have no independent output lines and adapting the values of an output schedule in any output line can lead to a modification of any other output schedule value in the whole graph.

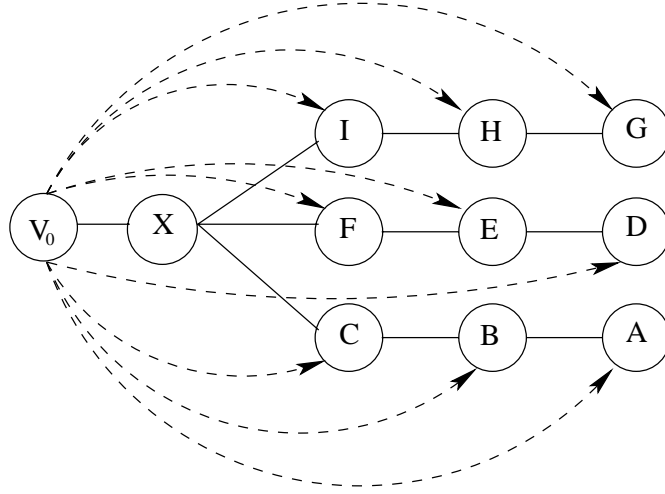


Figure 8.12: Constraint Graph with Connected Output Lines

We realized this experiment in the same way as the experiment previous, the only difference is the graph structure. We expected that adapting output schedules is slower than using a graph structure with independent output lines. However, adapting output schedules by using correction values should be faster than building a completely new output schedule by running the Bellmann and Ford Algorithm on the whole graph.

Figure 8.11 shows the results of this experiment. It is easy to see that the structure of the constraint graph has a huge influence on the efficiency of adapting output schedules by using correction values. However, in most cases building a completely new output schedule takes much more time than correct the existing schedule. This effect is emerged if the constraint graph is large. However, we can see the same effect as in the experiment above. If we modify large parts of a constraint graph, building a completely new output schedule is more efficient. It is noticeable that this effect is more distinctive than in the experiment above. The reason for this is that each single adaptation of an output schedule needs more time because it has influence on larger parts of the constraint graph.

8.4 Conclusion

The contribution of this chapter was to prove the practical usage of output constraints. Thus, it is important to evaluate qualitative properties considering modelling and usage of output constraints. So, we took the wave field synthesis as application scenario and determined how output constraints can be modelled and implemented for this scenario.

For wave field synthesis some requirements have been defined in chapter 2. The main points we had required were a synchronization which is defined relatively between output objects and checking it during data input and output.

For modelling output constraints we defined the required synchronizations by using a DDL definition as well as an XML structure. Furthermore, we transformed output constraints from their descriptive definition into a database internal representation. We used this representation for checking output constraints and producing a data output file.

Experiments have been used for achieving quantitative results considering the efficiency of the proposed methods for checking consistency and producing output schedules. Considering checking output constraints we can see, that the analytic approach is much faster than the Bellmann and Ford Algorithm. Nevertheless, we can check output consistency of large constraint graphs in less than a second.

The experiments dealing with producing output schedules showed that the structure of the constraint graph has high impact on the time required for adapting output schedules. We made an experiment to test the performance of building a new output schedule by adapting an existing one in the worst case. The results show that our algorithm performs quite well.

However, if we look at the absolute times, we see that checking output consistency and adapting output schedules needs more than a second in many cases. If we consider the situation that several data modifications are made in a transaction, we can adapt output schedules after each modification or at the end of the transaction. If we assume that usual constraint graphs deal with up to 250 output objects, we see that adapting output schedules after each modification is not critical. However, if we adapt the output schedule at the end of this transaction we will have a noticeable delay. Therefore, it is proposed to use save points after 25 or 50 modifications.

Chapter 9

Conclusion and Future Work

This thesis introduced a new kind of constraints for maintaining the semantics of multimedia data because the integrity concept of traditional database systems is not sufficient for maintaining the semantics of complex media data. We have named these new constraints ‘output constraints’ (figure 9.1) since they especially consider the correct data output of complex media data, like videos. This thesis provides all necessary prerequisites for integrating output constraints into a multimedia database system. We considered a constraint language for output constraints, the database internal representation of output constraints, methods for checking output constraints, and concepts for producing an output order of media data based on the defined output constraints.

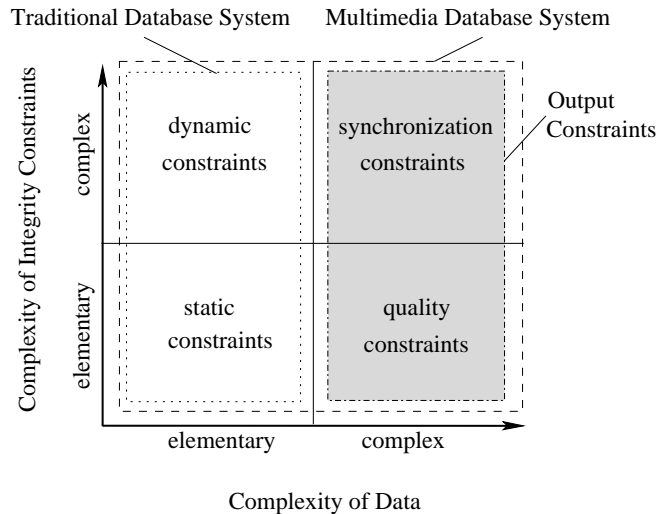


Figure 9.1: Classification of Integrity Constraints

Output constraints are an enlargement of the traditional integrity concept provided by database systems. A characteristic of traditional integrity constraints is their declarative definition based on a constraint language. Thus, for modelling output constraints we have introduced a constraint language which uses the same basics, i.e. first order

logic, as traditional constraint languages. Our constraint specification language must support temporal and spatial synchronization constraints. However, it is desired supporting both kinds of synchronization in almost the same manner. Therefore, we used Allen-Relations for defining temporal synchronization constraints as well as for defining spatial synchronization constraints. Originally, Allen-Relations define qualitative relationships between intervals. For output constraints we also need a quantitative definition of relationships between intervals. Thus, we have introduced restricted Allen-Relations that allow for a quantitative definition of relationships between intervals.

Allen-Relations are suitable for defining output constraints, but they do not support an efficient checking of output constraints. Thus, we need a representation of output constraints that makes efficient constraint checking possible. Difference constraints are a class of constraints that allows an very efficient checking. Therefore, we use difference constraints as database internal representation of output constraints. Thus, we have to introduced a transformation of our constraint language into difference constraints.

As methods for checking consistency of output constraints we used an approach on the basis of graph theory as well as an analytical approach. Both approaches require a constraint graph as data structure for working. Output constraints must be checked in the following cases: modifications on stored media data, modifications on output constraints, data output of media data. In order to avoid a delay during database transactions or data output we need an optimized constraint checking. The main idea for the optimization is to check only that part of the constraint graph that is affected by modifications or by data output. Therefore, we have developed a method that efficiently computes the ‘partial graph’.

For data output we need an output order that is adequate to the defined output constraints. This ‘output schedule’ can be produced based an the output constraints. While checking consistency by using the graph theoretical approach based on the Bellmann and Ford Algorithm an output schedule is computed. However, we have seen that an output schedule must be adapted in case of modifications on media objects or on Allen-Relations. We have proposed a method that allows an efficient adaptation of output schedules in case of modifications.

Our approach of output constrains has been evaluated on a practical application scenario. We implemented output constraints for supporting the wave field synthesis. By using output constraints for wave field synthesis we have achieved the following improvements:

- The modelling process becomes much easier because relations between sound sources can be defined in a relative and declarative manner.
- The database system checks consistency of output constraints after modifications on audio data. This is very important because audio data are often modified by different users,
- Based on output schedules, which were produced by output constraints, we have

built a special data organization of audio data. This data organization allows the data output of up to a hundred sound sources at the same time whereas without this data organization the system could only support 32 sound sources at the same time.

We made experiments for testing the performance of checking output constraints as well as for testing the performance of maintaining output schedules. These experiments have shown that the approaches for checking output consistency and for producing output schedules which are proposed in this thesis work very efficient. Thus, we can execute output constraints checking in database transactions of multimedia database systems without any noticeable delay of the transaction duration.

Future Work is required for a full integration of output constraints into database systems. Several aspects of output constraints are not considered in this thesis. However, we have developed some additional ideas which were not introduced in this thesis. We see a need of research especially for the following topics:

- Output constraints must be checked during the data output process. Especially, quality constraints must be checked permanently or in certain intervals. Thus, we need concepts for observing the data output stream. These mechanisms should be similar to triggers, but instead of data modifications they must observe output parameters, like the frame rate of a video. [Gul05] deals with this subject. In this work we have addressed the question of how a certain quality for video output can be guaranteed. We used output constraints for defining the desired video quality. Actually, we restricted the signal-to-noise ratio with an output constraint. Using this constraint we can deduce a constraint for the required bandwidth during the output. We stored a video with different resolutions in a database. The bandwidth was observed and an adequate resolution of the video was chosen. Each time the bandwidth is changed the video is adapted.

However, we can only observe one output parameter for videos. Thus, we need an generalization of this approach. The aim of this research is a general concept of triggering several properties of data streams during the data output process. These concepts must become parts of the database system.

- Another problem we have only considered partially is dealing with database queries. Output constraints are usually defined by the producer of media data. However, output constraints can also be defined in database queries by the user of the data. Our proposed approaches of checking output constraints and producing output schedules can be used in both cases. However, we need techniques for comparing the output constraints defined by the data producer with those defined by the data user. This is necessary if we want to check whether a database query is compatible with the output constraints defined by the data producer.

We have developed some ideas for checking compatibility between output constraints defined by the data producer and those that are included in a database

query. However, this idea only deals with output constraints considering synchronization. In a first pass we must check which media objects are involved in both sets of synchronization constraints. If we have media data which are involved in both sets, we must compute a solution (i.e. computing the output schedule) for the synchronization constraints that are defined in the database query. Now we have to determine whether this solution is also a solution for the synchronization constraints defined by the data producer. We can check this by applying the computed values into the difference constraints built by the other group of synchronization constraints.

- Output constraints should become part of the usual database concepts. Most commercial database systems support object-relational concepts, like class types. Thus, it is necessary to integrate output constraints into object-relational concepts. Usually, a class type defines a data structure in the structure schema and methods in the behaviour schema. However, there is no possibility for defining the data output. Therefore, we have introduced an output schema [Hei04a] which must be part of a class type. All output constraints concerning the attributes of the class type are defined in the output schema of the class type. So we can realize that each object in a class knows how it must be output.

However, this work is still on a very abstract level. Furthermore, it must be considered how output schemes can be used in database queries. At the moment only the conceptional level is considered. Further work is required to clarify how we can implement output schemes.

Appendix A

Harmless and Critical Allen-Relations

We assume for the following:

The intervals A and B have defined lengths c_A and c_B . This fact is leading to the following set of difference constraints: $\{end(A) - st(A) \leq c_A, st(A) - end(A) \leq -c_A, end(B) - st(B) \leq c_B, st(B) - end(B) \leq -c_B\}$

A meets B

Set of difference constraints:

$$\{end(A) - st(B) \leq 0, st(B) - end(A) \leq 0\}.$$

Constraint graph $G = \langle V, E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (end(A), st(B)), (st(B), end(A))\}$$

edge weights:

$$w(st(A), end(A)) = c_A, w(end(A), st(A)) = -c_A,$$

$$w(st(B), end(B)) = c_B, w(end(B), st(B)) = -c_B,$$

$$w(end(A), st(B)) = 0, w(st(B), end(A)) = 0$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$E_1 = \{(end(A), st(B))\}$$

$$E_2 = \{(st(B), end(A))\}$$

$$E_3 = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

It is easy to see, that E_1 and E_2 have an element, thus a cycle must exist. The weight of both edges is 0, therefore, the necessary condition is not fulfilled. This means the Allen-Relation A *meets* B cannot produce a negative cycle. Therefore, it is a harmless relation.

A starts B

Set of difference constraints:

$$\{st(A) - st(B) \leq 0, st(B) - st(A) \leq 0\}.$$

Constraint graph $G = \langle V, E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (st(A), st(B)), (st(B), st(A))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(st(A), st(B)) &= 0, w(st(B), st(A)) = 0 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$E_1 = \{(st(A), st(B))\}$$

$$E_2 = \{(st(B), st(A))\}$$

$$E_3 = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

It is easy to see, that E_1 and E_2 have an element, thus a cycle must exist. The weight of both edges is 0, therefore, the necessary condition is not fulfilled. This means the Allen-Relation A *starts* B cannot produce a negative cycle. Therefore, it is a harmless relation.

A finishes B

Set of difference constraints:

$$\{end(A) - end(B) \leq 0, end(B) - end(A) \leq 0\}.$$

Constraint graph $G = \langle V.E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (end(A), end(B)), (end(B), end(A))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(end(A), end(B)) &= 0, w(end(B), end(A)) = 0 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$E_1 = \{(end(A), end(B))\}$$

$$E_2 = \{(end(B), end(A))\}$$

$$E_3 = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

It is easy to see, that E_1 and E_2 have an element, thus a cycle must exist. The weight of both edges is 0, therefore, the necessary condition is not fulfilled. This means the Allen-Relation A finishes B cannot produce a negative cycle. Therefore, it is a harmless relation.

A equal B

Set of difference constraints:

$$\{end(A) - end(B) \leq 0, end(B) - end(A) \leq 0, st(A) - st(B) \leq 0, st(B) - st(A) \leq 0\}.$$

Constraint graph $G = \langle V.E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (end(A), end(B)), (end(B), end(A)), (st(A), st(B)), (st(B), st(A))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(end(A), end(B)) &= 0, w(end(B), end(A)) = 0, w(st(A), st(B)) = 0, w(st(B), st(A)) = 0 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$\begin{aligned} E_1 &= \{(end(A), end(B)), (st(A), st(B))\} \\ E_2 &= \{(end(B), end(A)), (st(B), st(A))\} \\ E_3 &= \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\} \end{aligned}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

It is easy to see, that E_1 and E_2 have an element, thus a cycle must exist. The weight of all these edges is 0, therefore, the necessary condition is not fulfilled. This means the Allen-Relation $A \text{ equal } B$ cannot produce a negative cycle. Therefore, it is a harmless relation.

A before(c_1) B

Set of difference constraints:

$$\{end(A) - st(B) \leq -c_1\}.$$

Constraint graph $G = \langle V, E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (end(A), st(B))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(st(B), end(A)) &= -c_1 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$\begin{aligned} E_1 &= \{\} \\ E_2 &= \{(st(B), end(A))\} \\ E_3 &= \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\} \end{aligned}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

It is easy to see, that E_1 has no element, thus a cycle cannot exist. This means the Allen-Relation $A \text{ before}(c_1) B$ cannot produce a negative cycle. Therefore, it is a harmless relation.

A before_V(c₁, l) B

Set of difference constraints:

$$\{end(A) - st(B) \leq -c_1, st(B) - end(A) \leq c_1 + l\}.$$

Constraint graph $G = \langle V, E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (end(A), st(B)), (st(B), end(A))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(end(A), st(B)) &= c_1 + l, w(st(B), end(A)) = -c_1 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$E_1 = \{(end(A), st(B))\}$$

$$E_2 = \{(st(B), end(A))\}$$

$$E_3 = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

The result of checking the necessary condition is that an edge exists in E_1 and another edge exists in E_2 . Thus, a cycle exist and because of the fact that the edge in E_1 has a negative weight a negative cycle is possible. Therefore, the relation $A \text{ before}_V(c, l) B$ is a critical relation and it has to be checked after a modification.

A overlaps(c₁, c₂) B

Set of difference constraints:

$$\{st(A) - st(B) \leq -c_1, end(A) - end(B) \leq -c_2, st(B) - end(A) \leq -1\}.$$

Constraint graph $G = \langle V, E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (st(B), st(A)), (end(B), end(A)), (end(A), st(B))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(st(B), st(A)) &= -c_1, w(end(B), end(A)) = -c_2, w(end(A), st(B)) = -1 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$\begin{aligned} E_1 &= \{(end(A), st(B))\} \\ E_2 &= \{(st(B), st(A)), (end(B), end(A))\} \\ E_3 &= \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\} \end{aligned}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

The result of checking the necessary condition is that an edge exists in E_1 and another edge exists in E_2 . Thus, a cycle exist and because of the fact that the edge in E_1 has a negative weight a negative cycle is possible. Therefore, the relation A overlaps(c_1, c_2) B is a critical relation and it has to be checked after a modification.

A overlaps_{V_s}(c_1, l, c_2) B

Set of difference constraints:

$$\{st(A) - st(B) \leq -c_1, end(A) - end(B) \leq -c_2, st(B) - end(A) \leq -1, st(B) - st(A) \leq c_1 + l\}.$$

Constraint graph $G = \langle V, E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (st(B), st(A)), (st(A), st(B)), (end(B), end(A)), (end(A), st(B))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(st(B), st(A)) &= -c_1, w(st(A), st(B)) = c_1 + l, \\ w(end(B), end(A)) &= -c_2, w(end(A), st(B)) = -1 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all

those edges that connect the start and end points of A or B .

$$\begin{aligned} E_1 &= \{(end(A), st(B)), (st(A), st(B))\} \\ E_2 &= \{(st(B), st(A)), (end(B), end(A))\} \\ E_3 &= \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\} \end{aligned}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

The result of checking the necessary condition is that an edge exists in E_1 and another edge exists in E_2 . Thus, a cycle exist and because of the fact that the edge in E_1 has a negative weight a negative cycle is possible. Therefore, the relation A overlaps $_{V_s}(c_1, c_2) B$ is a critical relation and it has to be checked after a modification.

A overlaps $_{V_e}(c_1, c_2, l) B$

Set of difference constraints:

$$\{st(A) - st(B) \leq -c_1, end(A) - end(B) \leq -c_2, st(B) - end(A) \leq -1, end(B) - end(A) \leq c_2 + l\}.$$

Constraint graph $G = \langle V, E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (st(B), st(A)), (end(A), end(B)), (end(B), end(A)), (end(A), st(B))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(st(B), st(A)) &= -c_1, w(end(A), end(B)) = c_2 + l, \\ w(end(B), end(A)) &= -c_2, w(end(A), st(B)) = -1 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$\begin{aligned} E_1 &= \{(end(A), st(B)), (end(A), end(B))\} \\ E_2 &= \{(st(B), st(A)), (end(B), end(A))\} \\ E_3 &= \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\} \end{aligned}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

The result of checking the necessary condition is that an edge exists in E_1 and another edge exists in E_2 . Thus, a cycle exist and because of the fact that the edge

in E_1 has a negative weight a negative cycle is possible. Therefore, the relation A overlaps $_{V_e}(c_1, c_2, l)$ B is a critical relation and it has to be checked after a modification.

A overlaps $_{V_{se}}(c_1, l, c_2, m)$ B

Set of difference constraints:

$$\{st(A) - st(B) \leq -c_1, st(B) - st(A) \leq c_1 + l, end(A) - end(B) \leq -c_2, st(B) - end(A) \leq -1, end(B) - end(A) \leq c_2 + m\}.$$

Constraint graph $G = \langle V, E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (st(B), st(A)), (st(A), st(B)), (end(A), end(B)), (end(B), end(A)), (end(A), st(B))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(st(B), st(A)) &= -c_1, w(st(A), st(B)) = c_1 + l, \\ w(end(A), end(B)) &= c_2 + m, w(end(B), end(A)) = -c_2, w(end(A), st(B)) = -1 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$\begin{aligned} E_1 &= \{(end(A), st(B)), (end(A), end(B)), (st(A), st(B))\} \\ E_2 &= \{(st(B), st(A)), (end(B), end(A))\} \\ E_3 &= \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\} \end{aligned}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

The result of checking the necessary condition is that an edge exists in E_1 and another edge exists in E_2 . Thus, a cycle exist and because of the fact that the edge in E_1 has a negative weight a negative cycle is possible. Therefore, the relation A overlaps $_{V_{se}}(c_1, l, c_2, m)$ B is a critical relation and it has to be checked after a modification.

A during (c_1, c_2) B

Set of difference constraints:

$$\{st(B) - st(A) \leq -c_1, end(A) - end(B) \leq -c_2\}.$$

Constraint graph $G = \langle V.E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (st(A), st(B)), (end(B), end(A))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(st(A), st(B)) &= -c_1, w(end(B), end(A)) = -c_2 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$\begin{aligned} E_1 &= \{(st(A), st(B))\} \\ E_2 &= \{(end(B), end(A))\} \\ E_3 &= \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\} \end{aligned}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

The result of checking the necessary condition is that an edge exists in E_1 and another edge exists in E_2 . Thus, a cycle exist and because of the fact that the edge in E_1 has a negative weight a negative cycle is possible. Therefore, the relation A during (c_1, c_2) B is a critical relation and it has to be checked after a modification.

A during_{V_s}(c₁, l, c₂) B

Set of difference constraints:

$$\{st(B) - st(A) \leq -c_1, st(A) - st(B) \leq c_1 + l, end(A) - end(B) \leq -c_2\}.$$

Constraint graph $G = \langle V.E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (st(A), st(B)), (st(B), st(A)), (end(B), end(A))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(st(A), st(B)) &= -c_1, w(st(B), st(A)) = c_1 + l, w(end(B), end(A)) = -c_2 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$\begin{aligned} E_1 &= \{(st(A), st(B))\} \\ E_2 &= \{(end(B), end(A)), (st(B), st(A))\} \\ E_3 &= \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B))\} \end{aligned}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

The result of checking the necessary condition is that an edge exists in E_1 and another edge exists in E_2 . Thus, a cycle exist and because of the fact that the edge in E_1 has a negative weight a negative cycle is possible. Therefore, the relation A during _{V_s} (c_1, l, c_2) B is a critical relation and it has to be checked after a modification.

A during _{V_e} (c_1, c_2, l) B

Set of difference constraints:

$$\{st(B) - st(A) \leq -c_1, end(B) - end(A) \leq c_2 + l, end(A) - end(B) \leq -c_2\}.$$

Constraint graph $G = \langle V, E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (st(A), st(B)), (end(A), end(B)), (end(B), end(A))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(st(A), st(B)) &= -c_1, w(end(A), end(B)) = c_2 + l, w(end(B), end(A)) = -c_2 \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$\begin{aligned} E_1 &= \{(st(A), st(B)), (end(A), end(B))\} \\ E_2 &= \{(end(B), end(A))\} \\ E_3 &= \{(st(A), end(A)), (end(A), end(A)), (st(B), end(B)), (end(B), st(B))\} \end{aligned}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

The result of checking the necessary condition is that an edge exists in E_1 and another edge exists in E_2 . Thus, a cycle exist and because of the fact that the edge

in E_1 has a negative weight a negative cycle is possible. Therefore, the relation A during $_{V_e}(c_1, c_2, l)$ B is a critical relation and it has to be checked after a modification.

A during $_{V_{se}}(c_1, l, c_2, m)$ B

Set of difference constraints:

$$\{st(B) - st(A) \leq -c_1, st(A) - st(B) \leq c_1 + l, end(B) - end(A) \leq c_2 + m, end(A) - end(B) \leq -c_2\}.$$

Constraint graph $G = \langle V, E \rangle$.

$$V = \{st(A), end(A), st(B), end(B)\}$$

$$E = \{(st(A), end(A)), (end(A), st(A)), (st(B), end(B)), (end(B), st(B)), (st(A), st(B)), (end(A), end(B)), (end(B), end(A)), (st(B), st(A))\}$$

edge weights:

$$\begin{aligned} w(st(A), end(A)) &= c_A, w(end(A), st(A)) = -c_A, \\ w(st(B), end(B)) &= c_B, w(end(B), st(B)) = -c_B, \\ w(st(A), st(B)) &= -c_1, w(end(A), end(B)) = c_2 + m, \\ w(end(B), end(A)) &= -c_2, w(st(B), st(A)) = c_1 + l \end{aligned}$$

It is possible to divide the set E in a set E_1 that contains all edges starting in output object A and in a set E_2 that contains all edges starting in output object B . E_3 has all those edges that connect the start and end points of A or B .

$$E_1 = \{(st(A), st(B)), (end(A), end(B))\}$$

$$E_2 = \{(end(B), end(A)), (st(B), st(A))\}$$

$$E_3 = \{(st(A), end(A)), (end(A), end(A)), (st(B), end(B)), (end(B), st(B))\}$$

Checking the necessary condition $\exists e_1 \in E_1 : \exists e_2 \in E_2 : w(e_1) < 0 \vee w(e_2) < 0$:

The result of checking the necessary condition is that an edge exists in E_1 and another edge exists in E_2 . Thus, a cycle exist and because of the fact that the edge in E_1 has a negative weight a negative cycle is possible. Therefore, the relation A during $_{V_{se}}(c_1, l, c_2, m)$ B is a critical relation and it has to be checked after a modification.

Appendix B

Analytical Constraints for Allen-Relations

For defining analytical constraints for restricted Allen-Relations the length of the output objects used in the Allen-Relations is required. In the following we denote the length of output object A with L_A and the length of output object B with L_B . We can compute both quantities as follows: $L_A = end(A) - st(A)$, $L_B = end(B) - st(B)$

Restricted Allen Relations	Analytical Constraints
$A \text{ equal } B$	$L_A = L_B$
$A \text{ during}(c_1, c_2) B$	$L_A \leq L_B - c_1 - c_2$
$A \text{ during}_{V_s}(c_1, l, c_2) B$	$L_A \leq L_B - c_1 - c_2$ $L_A \geq L_B - (c_1 + l) - c_2$
$A \text{ during}_{V_e}(c_1, c_2, m) B$	$L_A \leq L_B - c_1 - c_2$ $L_A \geq L_B - c_1 - (c_2 + m)$
$A \text{ during}_{V_{se}}(c_1, l, c_2, m) B$	$L_A \leq L_B - c_1 - c_2$ $L_A \geq L_B - (c_1 + l) - (c_2 + m)$
$A \text{ overlaps}(c_1, c_2) B$	$L_A \geq c_1 + 1$
$A \text{ overlaps}_{V_s}(c_1, l, c_2) B$	$L_A \geq c_1 + 1$ $L_A \leq L_B + c_1 + l - c_2$
$A \text{ overlaps}_{V_e}(c_1, c_2, m) B$	$L_B \geq c_2 + 1$ $L_B \leq L_A - c_1 + c_2 + m$
$A \text{ overlaps}_{V_{se}}(c_1, l, c_2, m) B$	$L_A \geq c_1 + 1$ $L_A \leq L_B + c_1 + l - c_2$ $L_B \geq c_2 + 1$ $L_B \leq L_A - c_1 + c_2 + m$
$A \text{ before}(c, l) B$	$c \leq (c + l)$

Table B.1: Analytical Constraints for Restricted Allen-Relation

Bibliography

- [ABC⁺76] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. Gray, P. P. Griffiths, W. F. King III, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson. System R: Relational Approach to Database Management. *ACM Trans. Database Syst.*, 1(2):97–137, 1976.
- [ABD⁺89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, pages 223–240, Kyoto, Japan, 1989.
- [Adi96] M. Adiba. STORM: An object-oriented Multimedia DBMS. In Nwosu et al. [NTB96], pages 47–84.
- [AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley Publishing Company, 1995.
- [All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- [All84] J.F. Allen. Towards a general Theory of Time and Action. *Artif. Intell.*, 23:123–154, 1984.
- [ASS00] S. Adali, M.L. Sapino, and V.S. Subrahmanian. An algebra for creating and querying multimedia presentations. *Multimedia Systems*, 8(3):212–230, 2000.
- [AvBF⁺92] P.M.G. Apers, C.A. v.d. Berg, J. Flokstra, P.W.P.J. Grefen, M.L. Kersten, and A.N. Wilschut. PRISMA/DB: A Parallel Mainmemory Relational DBMS. *IEEE Trans. Knowledge Data Engineering*, 4(6), 1992.
- [BBH⁺02] H. Berthold, F. Binkowski, A. Henrich, S. Hollfelder, W. Lindner, U. Merder, K. Meyer-Wegener, and G. Robert. Architektur Multimedialer Informationssysteme. *Informatik Forschung und Entwicklung*, 17:77–89, 2002.

- [BC98] E. Bertino and B. Catania. A Constraint-Based Approach to Shape Management in Multimedia Databases. *Multimedia Syst.*, 6(1):2–16, 1998.
- [BEH01] E. Bertino, A.K. Elmagarmid, and M.S. Hacid. Quality of Service in Multimedia Digital Libraries. *SIGMOD Record*, 30(1):35–40, 2001.
- [BEH03] E. Bertino, A.K. Elmagarmid, and M.S. Hacid. A Database Approach to Quality of Service Specification in Video Databases. *SIGMOD Record*, 32(1):35–40, 2003.
- [Ber02] H. Berthold. *A Federated Multimedia Database System*. PhD thesis, University of Dresden, 2002.
- [BF98] E. Bertino and E. Ferrari. Temporal Synchronization Models for Multimedia Data. *TKDE*, 10(4):612–631, 1998.
- [BGÖS97] J. L. Bruno, E. Gabber, B. Özden, and A. Silberschatz. Move-to-Rear List Scheduling: A New Scheduling Algorithm for Providing QoS Guarantees. In *ACM Multimedia*, pages 63–73, 1997.
- [BK99] S. Boll and W. Klas. ZYX - A Semantic Model for Multimedia Documents and Presentations. In *DS-8*, pages 189–209, 1999.
- [BKL96] S. Boll, W. Klas, and M. Lohr. Integrated database services for multimedia presentations, 1996.
- [BKW99] S. Boll, W. Klas, and U. Westermann. Exploiting ORDBMS Technology to Implement the ZYX Data Model for Multimedia Documents and Presentations. In *Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 232–250, 1999.
- [Bro78] M.L. Brodie. *Specification and Verification of Data Base Semantic Integrity*. PhD thesis, University of Toronto, 1978.
- [Bul01] D.C.A. Bulterman. SMIL 2.0, Part 1: Overview, Concepts and Structure. *IEEE MultiMedia*, 8(4):82–88, 2001.
- [BV94] M. Boone and E. Verheijen. The Wave Field Synthesis Concept Applied to Sound Reproduction. *AES Convention Paper presented at the 96th AES Convention Februar 1994, Amsterdam*, 1994.
- [BW00] E. Baralis and J. Widom. An algebraic approach to static analysis of active database rules. *ACM Transactions on Database Systems*, 25(3):269–332, 2000.
- [BZ93a] M.C. Buchanan and P.T. Zellweger. Automatic Temporal Layout Mechanisms. In *ACM Multimedia*, pages 341–350, 1993.

- [BZ93b] M.C. Buchanan and P.T. Zellweger. Automatically Generating Consistent Schedules for Multimedia Documents. *Journal of Multimedia Systems*, 1(2), 1993.
- [BZ93c] M.C. Buchanan and P.T. Zellweger. Scheduling Multimedia Documents Using Temporal Constraints. In P.V. Rangan, editor, *Network and Operating System Support for Digital Audio and Video*, pages 237–249. Springer-Verlag, 1993.
- [Cho92] J. Chomicki. History-less Checking of Dynamic Integrity Constraints. In *Proc. 8th IEEE International Conf on Data Engineering*, 1992.
- [Cho94] J. Chomicki. Temporal query languages: a survey. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic: ICTL'94*, volume 827, pages 506–534. Springer-Verlag, 1994.
- [CLR00] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press, 2000.
- [CLS00] K. S. Candan, E. Lemar, and V.S. Subrahmanian. View Management in multimedia databases. *The VLDB Journal*, 9(2):131–153, 2000.
- [Cod70] E.F. Codd. The Relation Model for Large Shared Data Banks. *Communication of the ACM*, 13(6):377–387, 1970.
- [Cod79] E.F. Codd. Extending the Database Relational Model to Capture more Meaning. In *ACM Trans. Database Syst.*, volume 4, 1979.
- [Con98] S. Conrad. A logic primer. In *Temporal Logic in Information Systems*, pages 5–30, 1998.
- [CP84] S. Ceri and G. Pelagatti. *Distributed Database Principles and Systems*. McGraw-Hill, Engelwood Cliffs, 1984.
- [CPM96] R. Cochrane, H. Pirahesh, and N. M. Mattos. Integrating Triggers and Declarative Constraints in SQL Database Systems. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 567–578, 1996.
- [CPS96] K.S. Candan, B. Prabhakaran, and V.S. Subrahmanian. CHIMP: A Framework for Supporting Distributed Multimedia Document Authoring and Presentation. In *ACM Multimedia*, pages 329–340, 1996.
- [CPS98] K.S. Candan, B. Prabhakaran, and V. S. Subrahmanian. Retrieval Schedules Based on Resource Availability and Flexible Presentation Specification. *ACM-Springer Multimedia Systems Journal*, 6(5):232–250, 1998.

- [CRS98] K. S. Candan, P.V. Rangan, and V. S. Subrahmanian. Collaborative Multimedia Systems: Synthesis of Media Objects. *IEEE Trans. Knowl. Data Eng.*, 10(3):433–457, 1998.
- [CT98] J. Chomicki and D. Toman. Temporal logic in information systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 3, pages 31–70. Kluwer Academic Publishers, Boston, 1998.
- [CW90] S. Ceri and J. Widom. Deriving Production Rules for Constraint Maintenance. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proceedings of the 16th VLDB Conference*, pages 566–577, Brisbane, Australia, 1990.
- [Dat81] C.J. Date. Referential integrity. In *Proc. 7th International Conf. on Very Large Data Bases*, 1981.
- [Dat83] C. J. Date:, editor. *An Introduction to Database Systems, Volume II*. Addison-Wesley, 1983.
- [dB89] E.O. de Brock. *De Grondslagen van Semantische Databases*. Academic Service, 1989.
- [Deß93] S. Deßloch. *Semantic Integrity in Advanced Database Management Systems*. PhD thesis, Universität Kaiserslautern, 1993.
- [DK95] A. Duda and C. Keramane. Structured temporal Composition of Multimedia Data. In *IW-MMDBMS*, pages 136–142, 1995.
- [EGPJ02] D.J. Ecklund, V. Goebel, T. Plagemann, and E.F. Ecklund Jr. Dynamic end-to-end QoS management middleware for distributed multimedia systems. *Multimedia Syst.*, 8(5):431–442, 2002.
- [ELG84] H.D. Ehrich, U.W. Lipeck, and M. Gogolla. Specification, Semantics and Enforcement of Dynamic Database Constraints. In *Proc. ACM SIGMOD International Conf. on the Management of data*, 1984.
- [eS97] M.A.P. e Silva. Dynamic integrity constraints definition and enforcement in databases: A classification framework. In *IICIS*, pages 65–87, 1997.
- [ES02] M. Erwig and M. Schneider. Spatio-Temporal Predicates. *IEEE Trans. Knowl. Data Eng.*, 14(4):881–901, 2002.
- [FGNS00] L. Forlizzi, R. Hartmut Güting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. In W. Chen, J.F. Naughton, and P.A. Bernstein, editors, *Proceedings of the 2000 ACM*

- SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 319–330. ACM, 2000.
- [GA91] P.W.P.J. Grefen and P.M.G. Apers. Integrity Constraint Enforcement Through Transaction Modification. In *Proc. 2nd International Conf. on Database and Expert Systems Applications*, 1991.
- [GA93] P.W.P.J. Grefen and P.M.G. Apers. Integrity control in relational database systems - An overview. *Data & Knowledge Engineering*, 10:187–223, 1993.
- [GIÖ98a] M. N. Garofalakis, Y. E. Ioannidis, and B. Özden. Resource scheduling for composite multimedia objects. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 74–85, 24–27 1998.
- [GIÖ98b] M.N. Garofalakis, Y.E. Ioannidis, and B. Özden. Resource Scheduling for Composite Multimedia Objects. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 74–85. Morgan Kaufmann, 1998.
- [GKK⁺03] D. Gabelaia, R. Kontchakov, A. Kurucz, F. Wolter, and M. Zakharyashev. On the Computational Complexity of Spatio-Temporal Logics. In I. Russell and S.M. Haller, editors, *Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference, St. Augustine, Florida, USA*, pages 460–464. AAAI Press, 2003.
- [GL97] M. Gertz and U.W. Lipeck. An Extensible Framework for Repairing Constraint Violations. In S. Jajodia, W. List, G.W. McGregor, and L. Strous, editors, *IICIS*, volume 109 of *IFIP Conference Proceedings*, pages 89–111. Chapman Hall, 1997.
- [Gol81] B.S. Goldstein. Constraints on Null Values in Relational Database Systems. In *Proc. 7th International Conf. on Very Large Data Bases*, 1981.
- [GÖS98] M. N. Garofalakis, B. Özden, and A. Silberschatz. On Periodic Resource scheduling for Continuous-Media Databases. *VLDB J.*, 7(4):206–225, 1998.
- [GP99] V. Goebel and T. Plagemann. Mapping User-Level QoS to System-Level QoS and Resources in a Distributed Lecture-on-Demand System. In *FTDCS*, pages 197–206. IEEE Computer Society, 1999.
- [Gre92] P.W.P.J. Grefen. *Integrity Control in Parallel Database Systems*. PhD thesis, University of Twente, 1992.

- [Gre93] P. Grefen. Combining theory and practice in integrity control: A declarative approach to the specification of a transaction modification subsystem. In *Proceedings of the 19th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Dublin, 1993*.
- [Gul05] C. Gulich. Überwachung von Ausgabequalität in einer Videodatenbank. Master's thesis, Technical University of Ilmenau, 2005.
- [GV89] G. Gardarin and P. Valduriez. *Relational Databases and Knowledge Bases*. Addison-Wesley, 1989.
- [Ham97] C.-J. Hamann. On the Quantitative Specification of Jitter Constrained Periodic Streams. In *MASCOTS'97, 1997*.
- [Hei04a] T. Heimrich. An Output Schema for Multimedia Data in Multimedia Database Systems. In *6th. Int. Baltic Conference on Databases and Information Systems*, pages 125–134, 2004. Riga, Latvia.
- [Hei04b] T. Heimrich. Output Constraints in Multimedia Database Systems. In S. Guler, A. G. Hauptmann, and A. Henrich, editors, *Proceedings MDDE '04, 4th International Workshop on Multimedia Data and Document Engineering, Washington, DC, USA, July 2nd 2004*. IEEE Computer Society, 2004.
- [Hei05] T. Heimrich. Modeling of Output Constraints in Multimedia Database Systems. In Y. P. Chen, editor, *11th International Conference on Multi Media Modeling (MMM 2005), 12-14 January 2005, Melbourne, Australia*, pages 399–404. IEEE Computer Society, 2005.
- [HK96] N.B. Hirzalla and A. Karmouch. A Multimedia Query Specification Language. In Nwosu et al. [NTB96], pages 160–183.
- [Hor85] W. Horak. Office Document Architecture and Office Document Interchange Formats: Current Status of International Standardization. *IEEE Computer*, 18(10):50–60, 1985.
- [HS90] F.G. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. In *NIST Hypertext Standardization Workshop, 1990*.
- [HS94] F. Halasz and M. Schwartz. The Dexter Hypermedia Model. *Commun. ACM*, 37(2):30–39, 1994.
- [HSRG05] T. Heimrich, K.U. Sattler, K. Reichelt, and G. Gatzsche. Verwaltung spatio-temporalen Audiodaten für die Wellenfeldsynthese. In G.Vossen, F. Leymann, P.C. Lockemann, and W. Stucky, editors, *BTW*, volume 65 of *LNI*, pages 444–453. GI, 2005.

- [JLR⁺98] M. Jourdan, N. Layaida, C. Roisin, L. Sabry-Ismaïl, and L. Tardif. Authoring Environment for Interactive Multimedia Documents. In *ACM Multimedia '98*, pages 267–272, 1998.
- [KA97] W. Klas and K. Aberer. Multimedia and its Impact on Database System Architectures. In *Multimedia Databases in Perspective*, pages 31–61. 1997.
- [KB96] S. Khoshafian and A.B. Baker. *MultiMedia and Imaging Databases*. Morgan Kaufmann Publishers, 1996.
- [KBL05] M. Kiefer, A. Bernstein, and M. Lewis. *Database Systems – An Application-Oriented Approach Models*. Pearson Education, Inc., 2005.
- [KC96] V.S. Subrahmanian K.S. Candan, B. Prabhakaran. CHIMP: A Framework for Supporting Distributed Multimedia Document Authoring and Presentation. In *Proc. of the ACM Multimedia Conference*, pages 329–340, 1996.
- [KMMW93] R. Käckenhoff, D. Merten, and K. Meyer-Wegener. Concept and Implementation of a Multimedia-Object Storage System. In T. Kirsche and H. Wedekind, editors, *Data Management for Advanced Applications*. 1993. Sonderforschungsbericht 182 ‘Multiprocessor- und Netzwerkkonfiguration’.
- [Krö88] G. Krönert. Genormte Austauschformate für Dokumente. *Informatik-Spektrum*, 11(2):71–84, 1988.
- [KS95] M.Y. Kim and J. Song. Multimedia Documents with Elastic Time. In *ACM Multimedia*, 1995.
- [LAE⁺04] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. X., and D. J. DeWitt. Limiting Disclosure in Hippocratic Databases. In *Proc. of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 108–119, 2004.
- [LÖS96] J.Z. Li, M.T. Özsu, and D. Szafron. Modeling of Video Spatial Relationships in an Object Oriented Database Management System. In *IW-MMDBMS*, pages 124–132, 1996.
- [LÖSO97] J.Z. Li, M.T. Özsu, D. Szafron, and V. Oria. MOQL: A Multimedia Object Query Language. In *The Third International Workshop on Multimedia Information Systems*, pages 19–28, 1997.
- [LSI96] N. LAYADA and L. SABRY-ISMAIL. Maintaining Temporal Consistency of Multimedia Documents Using Constraint Networks. In

- H. M. Vin M. Freeman, P. Jardetzky, editor, *Multimedia Computing and Networking*, pages 124–135, 1996.
- [Mar92] M. Marmann. *Datenmodellierungskonzepte fuer Hypermedien und ihre Abbildung auf Datenbanken*. PhD thesis, Distance University Hagen, 1992.
- [Mar03] U. Marder. *Multimedia-Metacomputing in Web-basierten multimedialen Informationssystemen*. PhD thesis, Universität Kaiserslautern, 2003.
- [MGSV99] M. Erwig, R.H. Gutting, M. Schneider, and M. Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *GeoInformatica*, 3(3):269–296, 1999.
- [MPS⁺00] I. Mirbel, B. Pernici, T.K. Sellis, S. Tserkezoglou, and M. Vazirgiannis. Checking the temporal integrity of interactive multimedia documents. *VLDB Journal: Very Large Data Bases*, 9(2):111–130, 2000.
- [MPV99] I. Mirbel, B. Pernici, and M. Vazirgiannis. Temporal integrity constraints in interactive multimedia documents. In *ICMCS, Vol. 2*, pages 867–871, 1999.
- [MR97] U. Marder and G. Robert. The KANGAROO Project – Enhancing a Media Server with Data Independence. In *Proc. of the 3th Int. Workshop on Multimedia Information Systems*, 1997.
- [MS96] S. Marcus and V. Subrahmanian. Foundations of Multimedia Database Systems. *Journal of the ACM*, 43(3):474–523, 1996.
- [MW03] K. Meyer-Wegener. *Multimediale Datenbanken – Einsatz von Datenbanktechniken für Multimedia-Systeme*. B.G. Teubner, 2. edition, 2003.
- [NTB96] K.C. Nwosu, B. Thuraisingham, and P.B. Berra, editors. *Multimedia Database Systems – Design and Implementation Strategies*. Kluwer Academic Publishers, 1996.
- [Ohm04] J.R. Ohm. *Multimedia Communication Technologie – Representation, Transmission and Identification of Multimedia Signals*. Springer, 2004.
- [OIÖ98] V. Oria, P. Iglinski, and M.T. Özsu. A Framework for Multimedia Database Systems. In *In Proc. 4th African Conference on Research in Computer Science, Dakar, Senegal*, pages 293–304, 1998.
- [OIS⁺97] M.T. Özsu, P. Iglinski, D. Szafron, S. El-Medani, and M. Junghanns. An Object-Oriented SGML/HyTime Compliant Multimedia Database Management System. In *ACM Multimedia*, pages 239–249, 1997.

- [Oom99] E. Oomoto. Integrity Constraints for Hyperlinks in a Hypermedia Database System: AYATORI. *IEICE Trans. Inf. & Syst.*, E82-D(1):165–179, 1999.
- [PTJ01] R. Price, N. Tryfona, and C.S. Jensen. Modeling Topological Constraints in Spatial Part-Whole Relationships. In Hideko S. Kunii, Sushil Jajodia, and Arne Sølvberg, editors, *ER*, volume 2224 of *Lecture Notes in Computer Science*, pages 27–40. Springer, 2001.
- [RSJM99] G. Ramalingam, J. Song, L. Joskowicz, and R.E. Miller. Solving Systems of Difference Constraints Incrementally. *Algorithmica*, 23(3):261–275, 1999.
- [Rus05] H. Rusch. Verwaltung von spatio-temporalen Audiodaten für die Wellenfeldsynthese. Master’s thesis, Technical University of Ilmenau, 2005.
- [SB00] G. Specht and M. Bauer. OMNIS/2: A Multimedia Meta System for existing Digital Libraries. In *Proc. of the 4th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2000)*, pages 180–189. Springer, LNCS 1923, 2000.
- [SBM99] M. Stonebraker, P. Brown, and D. Moore. *Object-Relational DBMS’s: Taking the Next Great Wave*. Morgan Kaufmann, 1999.
- [SMMW04] M. Suchomski, A. Märckz, and K. Meyer-Wgener. *Transformation of Knowledge, Information and Data: Theory and Applications*, chapter Multimedia Conversion with The Focus on Continuous Media. Information Science Publishing, 2004.
- [Sno00] R.T. Snodgrass, editor. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann Publishers, 2000.
- [Spe97] G. Specht. Complexity Analysis of Link Navigation in Dexter Based Hypermedia Database Systems. *Informatica*, 8(1):23–42, 1997.
- [Spe98] G. Specht. Multimedia-Datenbanksysteme: Modelle - Architektur - Retrieval. professorial dissertation, Technical University of Munich, 1998.
- [SST97] G. Saake, I. Schmitt, and C. Türker. *Objektdatenbanken - Konzepte, Sprachen, Architektur*. International Thomson Publishing, 1997.
- [Sto75] M. Stonbraker. Implementation of Integrity Constraints and Views by Query Modification. In *Proc. ACM SIGMOD International Conf. on the Management of Data*, 1975.

- [SV84] E. Simon and P. Valduriez. Design and Implementation of an Extendible Integrity Subsystem. In B. Yormark, editor, *SIGMOD'84, Proc. of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 9–17. ACM Press, 1984.
- [SWM95] R. Staehli, J. Walpole, and D. Maier. Quality of Service Specifications for Multimedia Presentations. *Multimedia Syst.*, 3(5-6):251–263, 1995.
- [TG01] C. Türker and M. Gertz. Semantic integrity support in sql:99 and commercial (object-)relational database management systems. *VLDB Journal: Very Large Data Bases*, 10(4):241–269, 2001.
- [Tür99] C. Türker. *Semantic Integrity Constraints in Federated Database Schemata*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, 1999.
- [Vaz96] M. Vazirgiannis. An object-oriented Modeling of Multimedia Database Objects and Applications. In Nwosu et al. [NTB96], pages 160–183.
- [VB97] M. Vazirgiannis and S. Boll. Events in interactive multimedia applications: Modeling and implementation design. In *International Conference on Multimedia Computing and Systems*, pages 244–251, 1997.
- [VK86] M. B. Vilain and H. A. Kautz. Constraint propagation algorithms for temporal reasoning. In *Fifth National Conference on Artificial Intelligence*, pages 377–382, 1986.
- [VKvBG95] A. Vogel, B. Kerhervé, G. v. Bochmann, and J. Gecsei. Distributed Multimedia and QOS: A Survey. *IEEE MultiMedia*, 2(2):10–19, 1995.
- [WBS00] R. Weber, K. Böhm, and H.J. Schek. Interactive-Time Similarity Search for Large Image Collections Using Parallel VA-Files. In *Research and Advanced Technology for Digital Libraries, 4th European Conference, ECDL 2000, Lisbon, Portugal, September 18-20, 2000, Proc.*, pages 83–92, 2000.
- [WGAK89] A.N. Wilschut, P.W.P.J. Grefen, P.M.G. Apers, and M.L. Kersten. Implementing PRISMA/DB in an OOPL. In *Proc. 6th International Workshop on Database Machines*, 1989.
- [WK87] D. Woelk and W. Kim. Multimedia Information Management in an Object-Oriented Database System. In P.M. Stocker, W. Kent, and P. Hammersley, editors, *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, Brighton, England*, pages 319–329. Morgan Kaufmann, 1987.

Thesen

1. Bisherige Integritätskonzepte in Datenbanksystemen sind nicht ausreichend für die semantisch korrekte Verwaltung von Mediendaten, wie z.B. Audiodaten. Das Problem existierender Ansätze liegt darin, dass die Semantik der Daten bei der Datenausgabe nicht überwacht wird. Durch fehlerhafte Synchronisation zwischen Mediendaten oder durch falsche Ausgabeparameter für Mediendaten können Fehler entstehen.
2. Multimedia Datenbanksysteme brauchen eine Erweiterung der bestehenden Integritätskonzepte. Wir führen in dieser Arbeit daher das Konzept der *Output Constraints* ein. Diese können vom Datenbankdesigner benutzt werden, um die semantisch korrekte Ausgabe von Mediendaten zu definieren.
3. Um Output Constraints definieren zu können, muss eine *Constraint Language* entwickelt werden. Diese Constraint Language muss auf der Prädikatenlogik aufbauen, aber auch die Definition von räumlichen und zeitlichen Bedingungen erlauben.
4. Die Output Constraints müssen vom Datenbanksystem effizient überprüft werden können. Daher ist eine datenbankinterne Darstellung von Output Constraints nötig, die eine effiziente Überprüfung unterstützt. Eine Abbildung der Constraint Language auf diese interne Repräsentationsform ist daher nötig.
5. Auf Grundlage der definierten Output Constraints müssen *Output Schedules* erzeugt werden. Diese definieren eine Ausgabeordnung für Mediendaten, die den Output Constraints entspricht. Das Datenbanksystem muss sicher stellen, dass die Mediendaten in dieser Ordnung ausgegeben werden.
6. Wenn Mediendaten in der Datenbank verändert werden oder vor der Ausgabe von Mediendaten, müssen Output Constraints überprüft und Output Schedules angepasst werden. Der von uns vorgestellte Algorithmus erlaubt es, diese Überprüfung und Anpassung ohne merklichen Zeitverlust durchzuführen.
7. Die von uns vorgestellten Output Constraints stellen eine erhebliche Verbesserung bei der Verwaltung von Mediendaten durch Multimedia Datenbanksysteme dar. Die Verbesserungen haben wir anhand eines praktischen Anwendungsszenarios (Wellenfeldsynthese) gezeigt.